NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

DESIGN AND IMPLEMENTATION OF WEB BASED
SUPPLY CENTERS MATERIAL REQUEST AND
TRACKING (SMART) SYSTEM USING WITH JAVA AND
JAVA SERVLETS

by

Cemalettin Ciftci

March 2001

Thesis Advisor :   C. Thomas Wu
Second Reader: Chris Eagle

Approved for public release; distribution is unlimited.

20010627 100

REPORT DOCUMENTATION PAGE		*Form Approved* *OMB No. 0704-0188*
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.		

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE March 2001	3. REPORT TYPE AND DATES COVERED Master's Thesis
4. TITLE AND SUBTITLE Design And Implementation Of Web Based Supply Centers Material Request And Tracking (Smart) System Using With Java And Java Servlets		**5. FUNDING NUMBERS**
6. AUTHOR Ciftci, Cemalettin		
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000		**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)		**10. SPONSORING / MONITORING AGENCY REPORT NUMBER**

11. SUPPLEMENTARY NOTES
The views expressed in this thesis are those of the authors and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.	12b. DISTRIBUTION CODE

13. ABSTRACT

In order for decision makers to efficiently make accurate decisions, pertinent information must be accessed easily and quickly. Component based architectures are suitable for creating today's three-tiered client–server systems. Experts in each particular field can develop each tier independently. The first tier can be built using HTML and web browsers. The middle tier can be implemented by using existing server side programming technologies that enable dynamic web page creation. The third tier maintains the database management systems.

Java servlets and Java provide programmers platform and operating system independent, multi-threaded, object oriented, secure and mobile means to create dynamic content on the web.

The Java Servlets Session Tracking API is a potential solution to problems arising from the fact that HTTP is a "stateless" protocol.

The use of connection pools with database applications provides faster data access, and decreases the use of system resources. Connection pools also offer a solution to the limited number of connections open to a specific database at a given time.

This thesis explores existing client-server architectures and server side programming technologies such as CGI, ASP and Java Servlets. The thesis also prescribes the design and implementation of a three-tier application using Java and Java servlets as the middle tier, and Java Database Connectivity to communicate with a database management system.

14. SUBJECT TERMS Software, Database, Structured Query Language (SQL), Common Gateway Interface (CGI), Active Server Pages (ASP), Java Database Connectivity (JDBC) and Java.			15. NUMBER OF PAGES 176
			16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFI-CATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL

NSN 7540-01-280-5500

Standard Form 298 (Rev.-89)
Prescribed by ANSI Std. 239-18 298-102

i

THIS PAGE INTENTIONALLY LEFT BLANK

DESIGN AND IMPLEMENTATION OF WEB BASED SUPPLY CENTERS MATERIAL REQUEST AND TRACKING (SMART) SYSTEM USING WITH JAVA AND JAVA SERVLETS

Cemalettin Ciftci
Lieutenant Junior Grade, Turkish Navy
B.S., Turkish Naval Academy, 1995

Submitted in partial fulfillment of the
requirements for the degree of

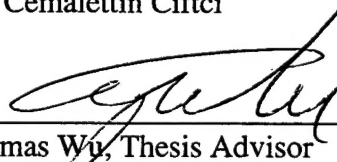**MASTER OF SCIENCE IN COMPUTER SCIENCE**

from the
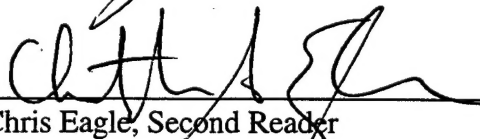
NAVAL POSTGRADUATE SCHOOL
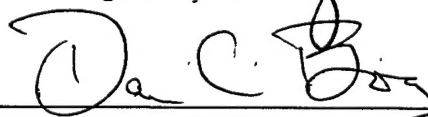**March 2001**

Author: _____
Cemalettin Ciftci

Approved by: _____
C. Thomas Wu, Thesis Advisor

_____
Chris Eagle, Second Reader

_____
Dan Boger, Chairman, Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

In order for decision makers to efficiently make accurate decisions, pertinent information must be accessed easily and quickly. Component based architectures are suitable for creating today's three-tiered client–server systems. Experts in each particular field can develop each tier independently. The first tier can be built using HTML and web browsers. The middle tier can be implemented by using existing server side programming technologies that enable dynamic web page creation. The third tier maintains the database management systems.

Java servlets and Java provide programmers platform and operating system independent, multi-threaded, object oriented, secure and mobile means to create dynamic content on the web.

The Java Servlets Session Tracking API is a potential solution to problems arising from the fact that HTTP is a "stateless" protocol.

The use of connection pools with database applications provides faster data access, and decreases the use of system resources. Connection pools also offer a solution to the limited number of connections open to a specific database at a given time.

This thesis explores existing client-server architectures and server side programming technologies such as CGI, ASP and Java Servlets. The thesis also prescribes the design and implementation of a three-tier application using Java and Java servlets as the middle tier, and Java Database Connectivity to communicate with a database management system.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

x

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. BACKGROUND

Supply Centers are expected to respond efficiently and accurately to the needs of military installations, especially during a war. In order for decision makers to make fast realistic decisions and to transfer material, supply centers need an effective system to process requisitions.

Currently, supply centers receive requisitions from the installations via wired messages or military mail depending on the urgency of the situation. The bulk of time lost while processing a request occurs while transferring the user request to the decision maker in the supply center.

Upgrading the legacy pograms used in supply centers and establishing a wide area network will enable messages to be transferred in a network environment. The implementation of an Enterprise Information System, a distributed architecture model that allows organizations to access and manipulate data over a network medium, is a good option for supply centers. The use of this system will save manpower and reduce the cost incurred while processing material requests.

The goal of this thesis is to choose the best architecture for the supply center system and to find the appropriate software technology to implement on designated architecture. Using the selected architecture and software technology a prototype for supply center request process will be designed and implemented. The primary research objective is to provide an object-oriented, web-based data retrieval system for the supply centers. The optimal solution would be independent of a specific Operating System and easy to develop and modify.

B.   SCOPE

The scope of this thesis is to design and implement an object oriented, component based enterprise information system for supply centers to process material requests. In the first phase of thesis research various client-server architectures will be reviewed to find the appropriate model for the structure of supply center material request processing system. Then existing software technologies used to create dynamic request response systems will be examined. Java Servlets that were found to best meet the Navy's expectations, will be further discussed and analyzed.

In order to enable request processing program, which will be implemented during the thesis research, to access supply center database, Java Database Connectivity (JDBC), a technology that provides Java programs database access, will be studied and used in the implementation.

Once the background information is gathered, the prototype will be constructed using the prescribed technologies to provide a solution that is cost-effective, fast, scalable and operating system independent.

C.   ORGANIZATION OF THESIS

This thesis will consist of the following chapters and given topics:

Chapter I will provide an overview of the existing system, the proposed system, and lay out the structure of the thesis.

Chapter II: Enterprise System Architectures: This chapter will provide the study of existing client-server architectures to develop an enterprise system. Each method will be studied in detail to make the best selection for the proposed system architecture.

Chapter III: Server Side Programming: This chapter will explore existing techniques used to create dynamic web pages, which enables interaction and data exchange between the users and database server.

Chapter IV: Database Access Methods: Database Access Methods, including native drivers, Open Database Connectivity(ODBC) and Structured Query Language(SQL) will be discussed to learn the ways to access databases.

Chapter V: Java Database Connectivity: The JDBC package of Sun Micro System's Java provides the web developers concise and efficient ways to access data in Relational Databases (RDBMS). In this chapter JDBC Application Programming Interface (API) will be studied to give the reader the basic knowledge required to understand the implementation.

Chapter VI: Implementation of a Web-based Client-Server System using Java Servlets and JDBC: This chapter will basically provide detailed discussion on the prototype's implementation.

Chapter VII: Conclusion: This chapter will discuss the benefites of the proposed system and potential improvements for the future.

THIS PAGE INTENTIONALLY LEFT BLANK

II. CLIENT / SERVER ARCHITECTURES

The term client/server dates back to the 1980s and refers to personnel computers (PCs) on a network. The client-server architecture is intended to improve usability, flexibility, interoperability and scalability of enterprise systems. A client is a computer which requests a service, and a server is the one who provides and delivers the requested service. A single computer, depending on the configuration, can also be both client and server. Client server architectures are used throughout industry and the military, which provides versatile networking infrastructure that supports insertion of new technology to the system.

R.Harkey Orfali defines the client/server model as "The client/ server model, entails two autonomous processes working together usually over a network; client processes request specific services which server processes respond to and process [Ref1]".

With the installation of the networking environment, client and server have the capability to communicate with each other.

Client-server system architectures can be grouped in four different types, which are one, two, three and N-tiered. All of these client-server system architectures have the following characteristics in common [Ref.1].

- Service: Client/server is primarily a relationship between processes running on separate or possibly the same machine. The server is the provider of services. The client is a consumer of services. In essence, the concept of client/server roles provide a clean seperation of function based on the idea of service.

- Shared Resources: A server can service many clients at the same time and regulate their access to shared resources.

- Asymmetric protocols: Clients always iniate a dialog by requesting a service. Servers passively await requests from the clients.

- Transparency of location: The server is a process that can reside on the same machine as the client or separate one.

To distinguish various architectures, each architecture will be discussed in detail

A.   ONE-TIER ARCHITECTURES

One-tier architecture is a self contained client and server. Since there is only one architectural layer the user interface, data processing and the data itself co-exist in one place. The best example of this architecture is legacy main frames, which basically consist of one mainframe computer and numerous dumb terminals which do not have their own memory and storage area.

Since all system resources are controlled by the mainframe the data processing handled by the mainframe. Whenever a business rule, which is part of the application logic, changes the administrators do not have to update each terminal. When the application containing the business rules is updated on the mainframe, all the clients will use the same new application information. In one tiered systems, user interface, data processing and data storage can not be separated from each other. This drawback lacks the component based arcitecture and componet reuse at a later time. A representation of one tiered architecture can be seen in the following figure.
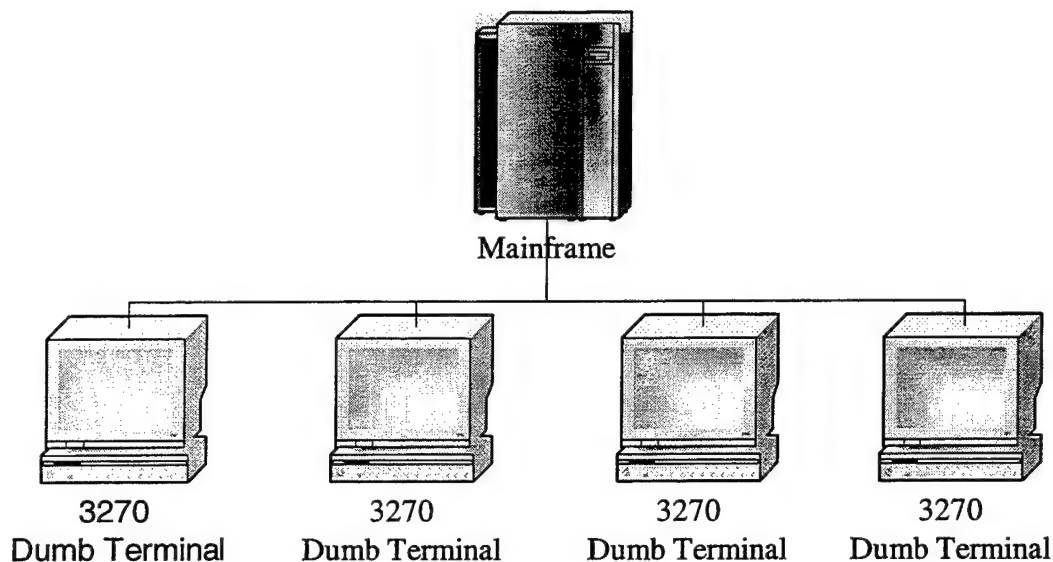
Figure 2.1 One-Tier Architecture

The current system architecture of most supply centers today fits into the one tiered architecture. One tiered architecture lacks the functionality, extensibility, scalability and flexibility of the current material management and request processing system. Given the ongoing system updates and installation of networking equipment, the ideal architecture for supply center request processes are 2 or more tiered architectures.

B.   TWO-TIER ARCHITECTURES

Two-tier architectures consist of three components distributed over two layers: the client, requester of a specific service and, the server, provider of the requested service. The three components of the system can be listed as follows:

- User system interface

- Processsing management

- Database management

7

In two tier architecture, user system interface is exclusively allocated to the client. As a result, the server would be responsible for database management and the process management would be split between the server and the client. Since the client will be responsible for process management as much as the server and will also be running the user interface, this architecture requires powerful client machines compared to one-tier architectures. Simply stated, the user system interface on the client side invokes the services from the database management server.

In two-tier architectures, most of the application work is loaded on the client . The server which is mainly responsible for database management, provides access to data which is needed by the client application.

First Tier

Client

User Interface

Tasks Services

- User Interface
- Presentation Services
- Application Services

Second Tier

Database Server

Database

Database Server

Tasks Services

- Application Services
- Business Services
- Data Services
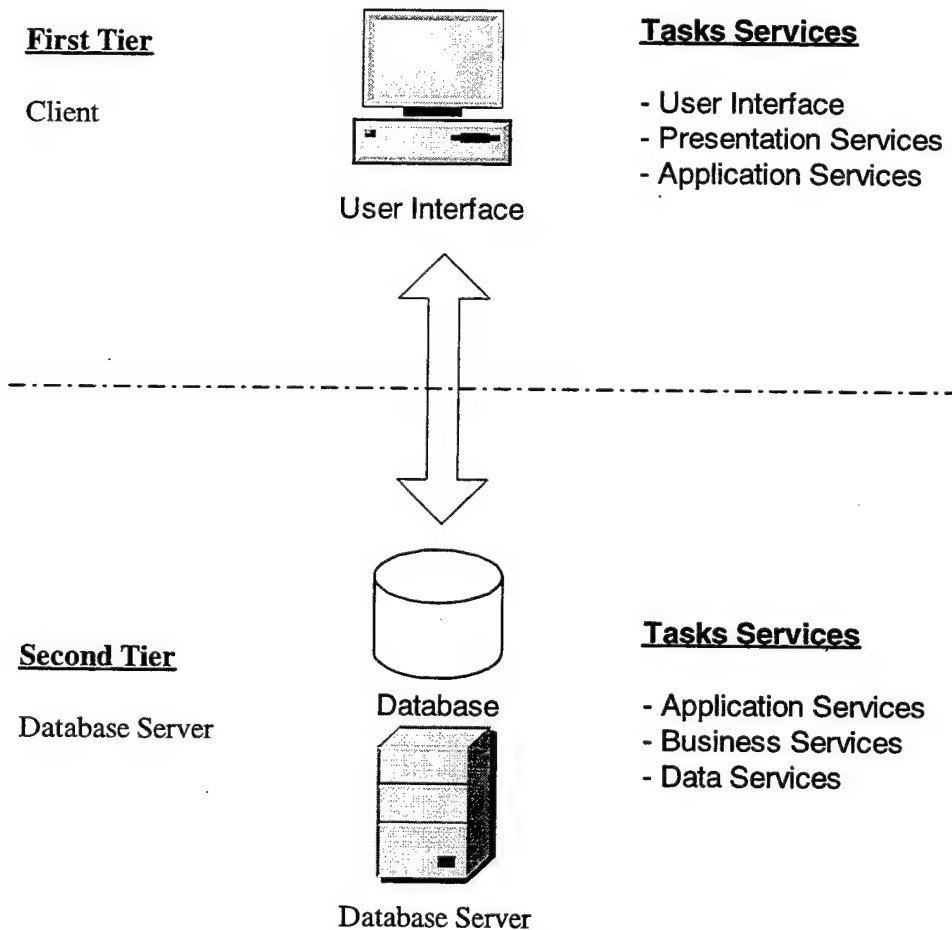
Figure 2.2 Two-Tier Architecture

The problems associated with two-tier architectures arise from the application logic located on the client side. If the logic changes, then changes must be made to all of the clients accessing the server. This creates an extra burden for administrators and makes the update process error prone. The clients that fail to be updated, will get incorrect information or in some cases never get access to information on the server.

Another problem is the limited technical capabilities of the client arise from the system configuration. Since the data to be analyzed and processed must be transmitted from the server and stored at the client machine, the data scalability is limited by the capability of the client machine which is related to the system configuration such as, system memory, storage are(i.e Random Access Memory (RAM), Harddisk Drive (HDD)). As a result of these limitations, it would be more beneficial to use the server's high processing capabilities.

C.   THREE-TIER ARCHITECTURES

Three-tier architectures have basically been introduced to overcome the limitations of the two-tier applications previously discussed, such as the number of connections to the server. The third tier (middle tier server) is added between the user interface (client) and the data management (server) components to provide process management services where program logic and business rules are executed. Thus the server can accommodate hundreds of users (as compared to the two tier applications) by providing functions such as queuing, application execution, and database staging.

The three tier architecture is used when an effective distributed client-server design is needed that provides (when compared to the two-tier) increased performance, flexibility, maintainability, reusability, and scalability, while hiding the complexity of distributed processing from the user. Spreading functions among multiple tiers provides scalability, faster performance, robustness and flexibility. By implementing a middle tier, data can be integrated from multiple servers.

9

The third tier provides database management functionality, and is dedicated to data and file services which can be optimized without using any proprietary database management system languages. As a result programmers have wide range of programming languages (as compared to one specific to database management system vendor) to optimize the database. The database management component ensures that the data is consistent throughout the distributed environment by using features such as data locking, consistency, and replication. Since data consistency and replication will be provided by the database management system, programmers will not be required to write their own code to handle these processes. It is possible that connectivity between tiers can be dynamically changed depending upon the user's request for data and services. As a result less bandwidth (compared to one and two-tier architectures) for connecting the tiers will be required and efficiently used.

The middle tier provides process management services (such as process development, process enactment, process monitoring, and process resourcing) which are shared by multiple applications.

The middle tier server (also referred to as the application server) improves performance, flexibility, maintainability, reusability, and scalability by centralizing process logic. Centralized process logic makes administration and change management easier by localizing system functionality so that changes must only be written once, and placed on the middle tier server to be available throughout the system. With other architectural designs, a change to a function (service) would need to be written into every application [Ref.3].

In addition, the middle process management tier controls transactions and asynchronous queuing to ensure reliable completion of transactions [Ref.4]. The middle tier manages distributed database integrity by the two phase commit process. The middle provides access to resources based on names instead of locations, and thereby improves scalability and flexibility as new components are added or moved .

When the middle tier is divided in two or more units with different functions, the architecture is often defined as multi-tier. This is the case, for example, of some Internet

applications. These applications typically have light clients written in HTML, and application servers written in C++ or Java. The gap between these two layers is too big to make linking them together possible. Instead, there is an intermediate layer (web server) which receives requests from the Internet clients and generates html using the services provided by the application logic (business layer). This additional layer provides further isolation between the application layout and the application logic.

Three-tier architecture facilitates software development because each tier can be built and executed on a separate platform, thus making it easier to organize the implementation of each tier component. Also, three tier architecture readily allows different tiers to be developed in different languages, such as a graphical user interface language or light internet clients (HTML, applets) for the top tier; C, C++, SmallTalk, Basic, or Ada for the middle tier; and SQL for much of the database tier.

First Tier

Client

Tasks Services

User Interface
Presentation Services

User Interface

Second Tier

Application Server

Business
Objec/Component

Tasks Services

Application Services
Business Services

Third Tier

Database Server

Database

Database Server

Tasks Services

Data Services
Data Validation
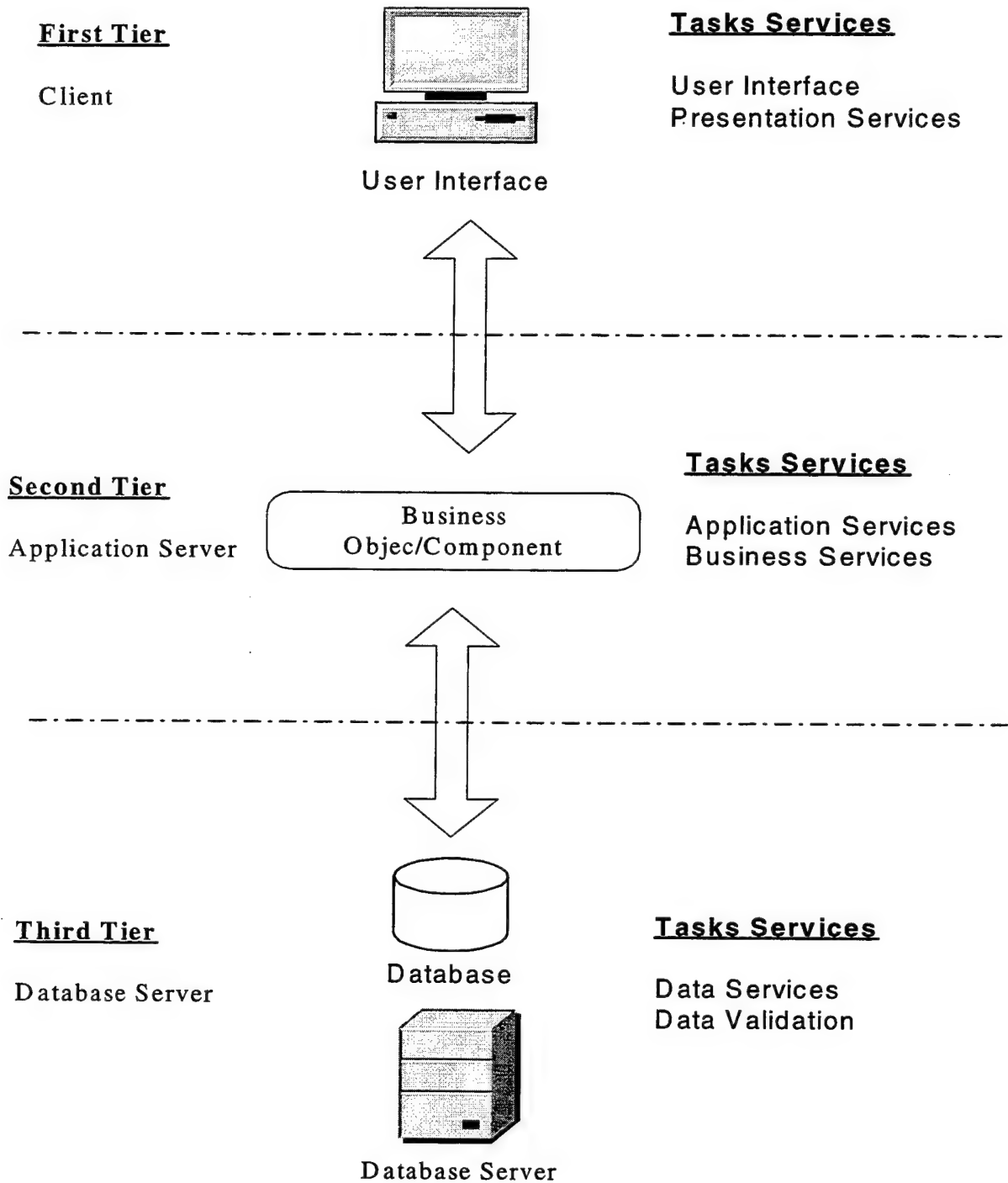
Figure 2.3 Three-Tier Architecture

12

D.   CONCLUSION

Three tier architecture seems to best meet the needs of todays Supply Centers. Using the three-tier architecture model, each component of the request processing system can be built separately. This feature brings scalability to the system, in which each part of the system could be developed by experts in each particular field. For example, the user interfaces(which solely consist of HTML content) can be built by webmasters. The database can be planned and implemented by the database administers who have strong knowledge of Structured Query Language (SQL). The middleware applications are left to the programmers whose main job would be to implement the business rules and interface between the data server and user interface.

Another benefit of the three-tier architecture is that more secure systems could be developed. Since the system is divided into parts which can be developed separately, each component can use a unique communication protocol independent of the other protocols. As seen in the following figure, the TCP/IP protocol suite can be used between the web server and web clients, while named-pipes are used as the communication protocol between web server and database server. Even if an unwanted user gains access to the web server, he would not be able to get direct access to the database server and confidential data .

It is also possible to integrate extra security features between the web server and client browsers, such as a fire wall. The database server would be oblivious to the presence of the fire wall and never need to know of its existence.

User Interface

Web Server

Database Server

TCP/IP

Named Pipes

Database

First Tier

Second Tier

Third Tier
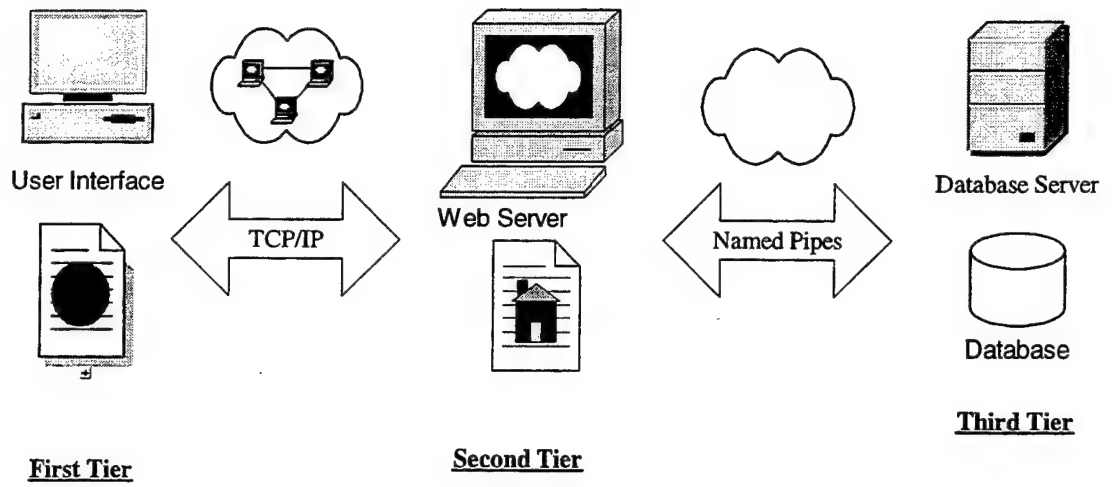
Figure 2.4 Use of Multiple Protocols in Three-Tier Architecture

14

III.   SERVER SIDE PROGRAMMING

If a company or individual desires to implement truly interactive, dynamically created web pages to handle user requests, then client-side scripting such as JavaScript and VBScript alone are not sufficient. Since client side scripts lack the ability to communicate with database servers and create dynamic pages depending on the stored data the programmers should look for something other than scripting languages working on the client side which would work on the server and create the desired dynamic pages. There are a few existing technologies whose purpose is to implement server side programming which are reviewed in the following sections. Each server side programming technology has certain strengths as well as drawbacks which have been discovered during the time of study.

A.   COMMON GATEWAY INTERFACE (CGI)

Common Gateway Interface(CGI) is the most well known and widely used application interface on the Internet. The reason for its popularity is that CGI is the first method to implement server side programming. CGI enables HTTP clients (i.e.s. web browsers) to communiciate with programs across the network through the web server. In spite of the fact that there is no specific programming language required to develop CGI applications, Perl is the first one that comes to mind for its popularity and wide acceptance. The following table lists of programming languages which may be used to write CGI programs.

Compiled Languages	Interpreted Languages
C	Perl
C++	TCL
Ada	Unix

Table 3.1 CGI Programming Languages

15

A web browser obtains information from a user using some kind of HTML form, (e.g. option boxes, buttons etc.) and sends that information to a web server. Depending on the request made by the user a CGI program, perhaps written in perl, is executed. The information gathered from the user is passed to the program using the CGI interface. The result of the program is sent to the user in HTML form which could also contain plug-ins such as video and sound. The interaction between the server and client is depicted in the following figure.

Client
Web Browser

Web Server

Application

CGI
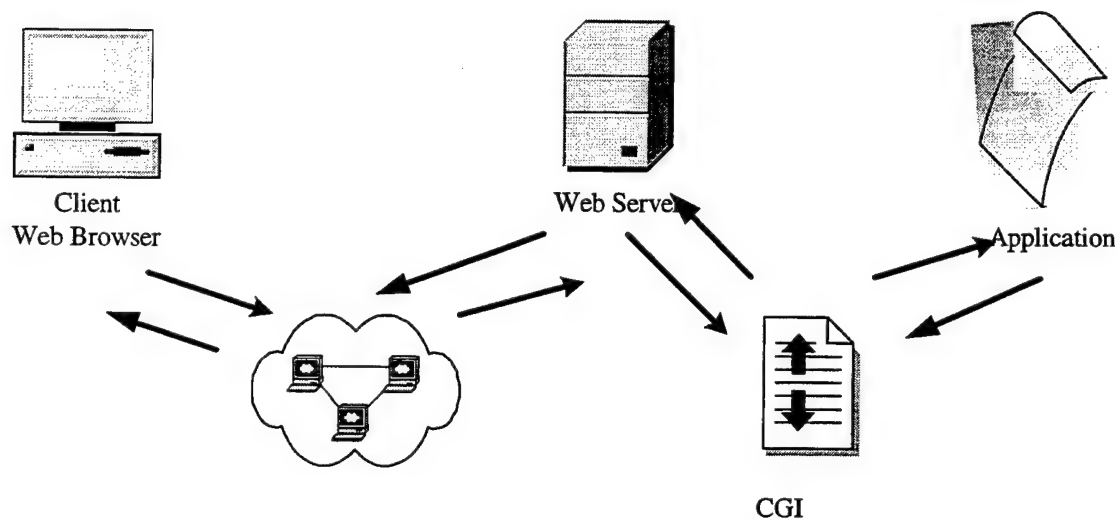
Figure 3.1 Typical Path of a CGI Based Application

It is possible to update enterprise databases on the Internet without knowing where the server is located. Using CGI technology it becomes possible for the people shop online.

1.   CGI-Perl Example

A simple program will be presented in the next section for the purpose of gaining an understanding of CGI programming. This program connects to an existing database, and processes the SQL query entered by the user.

The web page presented below, calls the perl program which is going to be executed upon the user request. The full code for the web page and perl code are provided in the appendices.

a.   HTML File

An HTML form is created to get SQL statements from the user which is by default "SELECT * FROM REQUESTS". When the user clicks the Send Query button, the action (defined in the ACTION parameter of the form), takes place and the data.pl program located in cgi-bin directory is executed.

```
<FORM METHOD = "POST" ACTION = "cgi-bin/data.pl">
    <INPUT TYPE = "TEXT" NAME = "QUERY" SIZE = 40
        VALUE = "SELECT * FROM REQUESTS"><BR><BR>
    <INPUT TYPE = "SUBMIT" VALUE = "Send Query">
</FORM>
```

b.   Perl File

In the next phase, the perl code which access the database and submits the query will be reviewed. Note that the Win32::ODBC module will be used by the CGI program to provide database interaction functionality.

```
use Win32::ODBC;
```

The HTML name of the text field, QUERY is passed to *param()* function which returns the user input string *$querystring*.

```
$querystring = param(QUERY);
```

Another scalar variable *$DSN* is created and assigned, *"THESISSQL"* ,(i.e. the ODBC Data Source Name of the database used for the example.)

```
$DSN = "THESISSQL";
```

By passing the Data Source Name DSN to the constructor for Win32::ODBC, a connection named Data is created to the database.

```
if (!($Data = new Win32::ODBC($DSN))){
```

17

If there is an error connecting to the database it is reported to the user.

print "Error connecting to " .$DSN. "\n";

print "Error: " . Win32::ODBC::Error() . "\n";

exit;}

The record set generated by the SQL statement is passed to the connection variable *$Data*.

if ($Data->Sql($querystring))

{ print "SQL failed.\n";

print "Error: " . $Data->Error() . "\n";

$Data->Close();

exit;}

Prepare the ouput HTML page.

print "<BODY BACKGROUND = \"/images/back.gif\">";

print "<BASEFONT FACE = \"ARIAL,SANS-SERIF\" SIZE = 3>";

print "<FONT COLOR = BLUE SIZE = 4> Search Results </FONT>";

A scalar counter variable is created to store the total number of records.

$counter = 0;

Create a table to print the results.

print "<TABLE BORDER = 0 CELLPADDING = 5 CELLSPACING = 0>";

As long as data is returned from method *FetchRow()*, *DataHash* is used to retrieve the fields in a row from the recordset and data is placed in variable *%Data*. An array of keys each of which is associated a value in the array values, is created. The variable *%Data* is passed to the function *keys()* and returns all the keys associated with hash function.

while($Data->FetchRow()){

%Data = $Data->DataHash();

@key_entries = keys(%Data);

print "<TR>";

foreach loop takes one of the records and divides it into corresponding fields.

foreach $key(keys(%Data)) {

The following line indicates the beginning of a field in a table.

print "<TD BGCOLOR = #9999CC>$Data{$key}</TD>";   }

After printing all the fields in a row, the current row is closed,

print "</TR>";

and the total number of results is increased.

$counter++;}

When all the rows in the table have been printed the table is closed.

print "</TABLE>";

HTML tag is also closed.

print end_html;

The connection to the database is terminated.

$Data->Close();
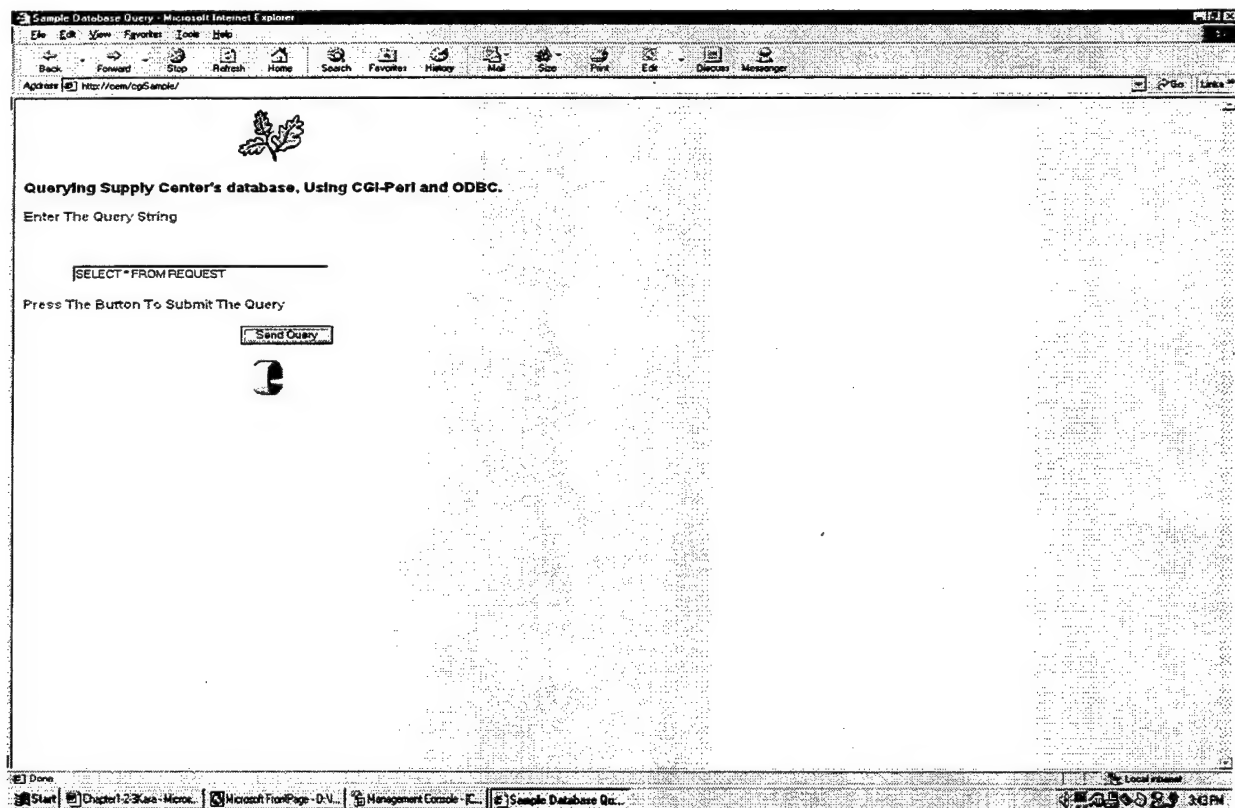
Figure 3.2 CGI Sample Main Page

19
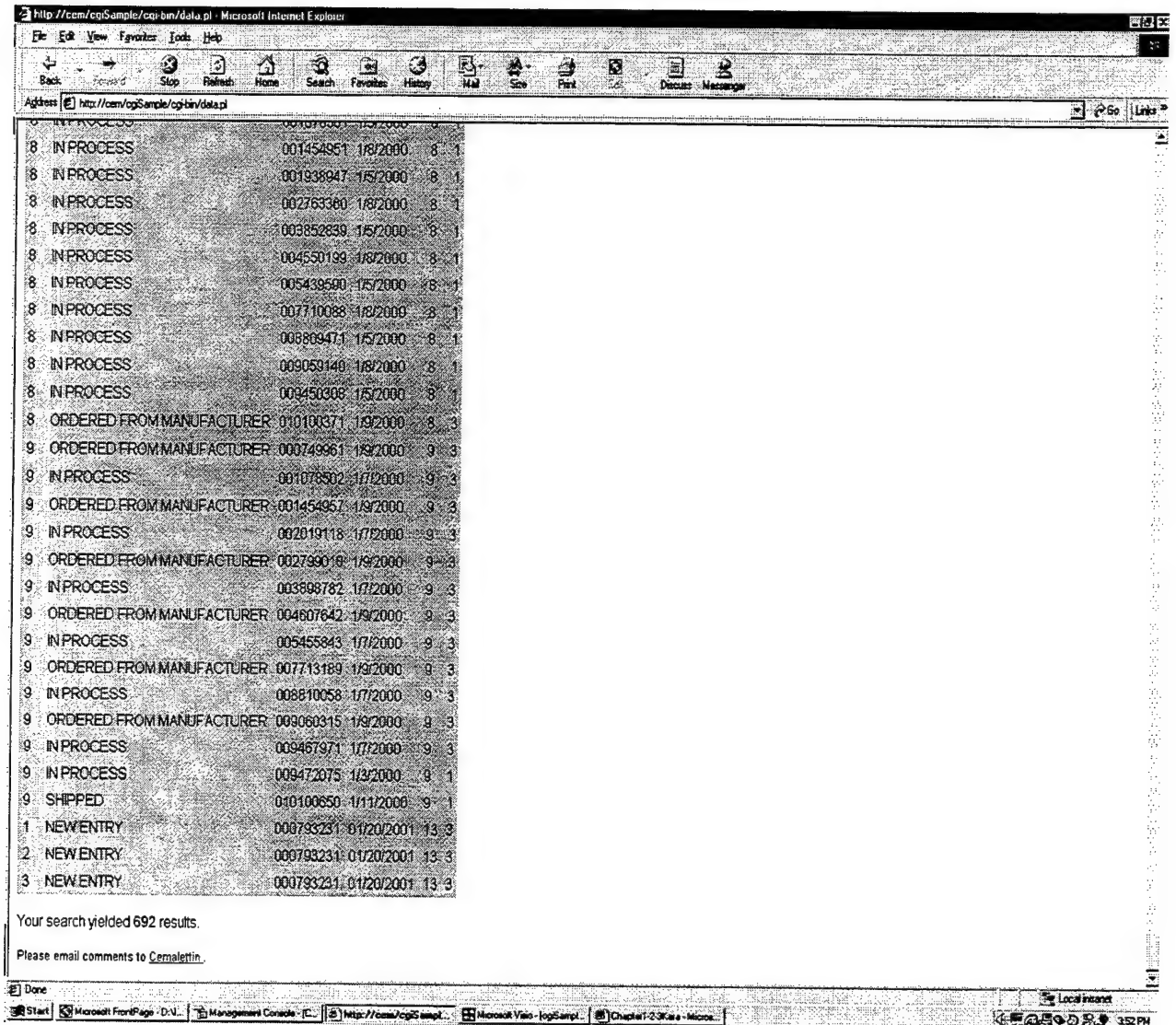
Figure 3.3 CGI Sample Results Page

C.   ACTIVE SERVER PAGES (ASP)

Active Server Pages are a Microsoft technology for sending dynamic web content, which includes HTML, client side scripts and Java applets, to the client. ASP pages are basically text files that are processed in response to a client's request. An ASP file has an extension of ".asp", and contains HTML tags and scripting code.

Although another scripting language such as JavaScript can be used for ASP scripting, VBScript is the default language for ASP, unless specifically defined elsewhere in the code. VBScript is a subclass of Microsoft's event driven, partially object oriented language of Visual Basic. Visual Basic is a compenent of Visual Studio Development suite distributed by Microsoft.

Active Server Pages were initially been developed to work with Microsoft's web server Internet Information Server (IIS) to enable IIS to serve dynamic content. Microsoft's Personel Web Server which is a version of IIS that runs on client machines other than the Windos NT Server Operating System, has also been enabled to work with Active Server Pages .

Third party developers have developed software, allowing ASP to be used on operating systems other than Windows. Chilisoft is one of these third party companies which presents programs for almost all known Commercial Of the Shelf (COTS) operating systems.

1.   Active Server Pages Object Model

According to Greg Buczek in his book Instant ASP Scripts, [Ref2] ASP Object model defines five primary objects: the Request object, the Response object, the Application object, the Server object and the Session object. The purpose of these object are found in the following table.

Request object	Supplies information from the user.
Response object	Contains methods and properties for building a response to the user.
Application object	Deals with properties that govern a grouping of Web pages referred to as an application.
Server object	Deals with the creation of server components and server settings.
Session object	Contains methods and properties pertaining to a particular visitor.

Table 3.1 Active Server Pages Objects

2.   How Active Server Pages Work

The steps of running an ASP page are as follows:

- User enters the address of the ASP file into the address bar of web browser.
- Browser sends a request for the ASP file to the web server.
- The web server recognizes the request for an ASP page.
- The web server retrieves the ASP file from disk or memory.
- The web server sends the file to theASP engine.
- Active Server Page are processed from top to bottom as scripting code encountered is executed. If a result needs to be produced, that result is . produced as an HTML file.
- The resulting HTML file is sent back to client browser.
- The HTML file including the result is interpreted by the web browser and is displayed.
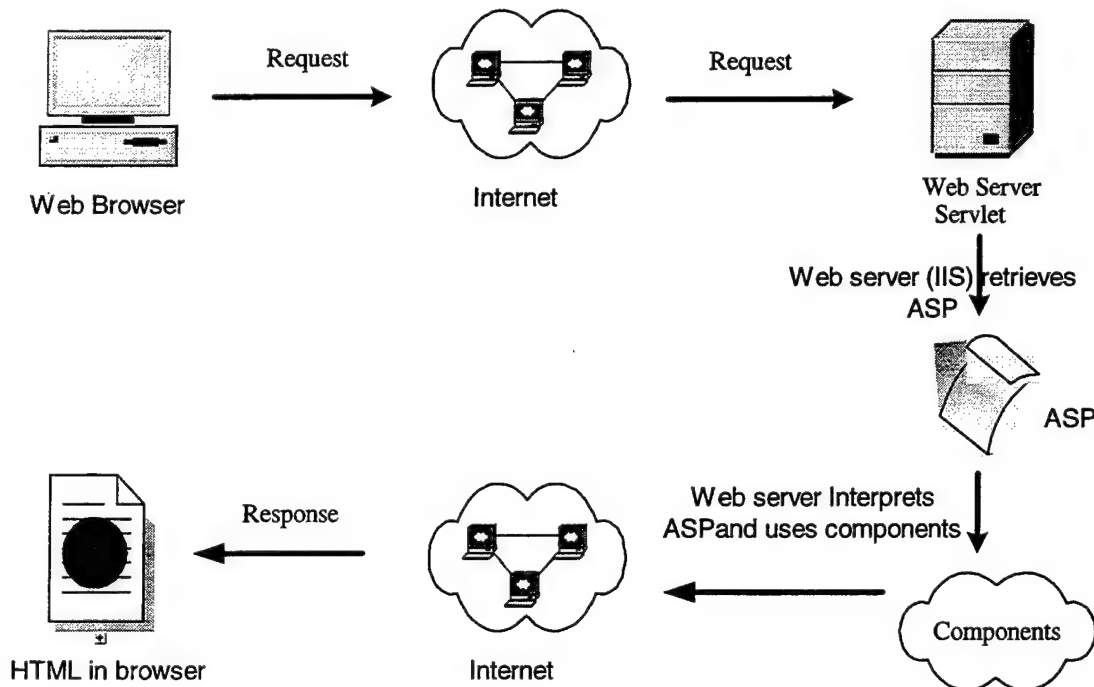
Figure 3.4 ASP Request Response Flow Diagram

As an example, consider the following path. A login page that retrieves the usernames from a database and shows the user this information in the form of a drop down menu and asks for a password. If the session object already has userId or the cookie in the user computer has an ID, the existing user is prompted. When the password is entered by the user, another ASP file is called to check the password. The password is compared with the one in the current database. If the password does not match, the user request is discarded. In case of a match, the next page in processing the user request is returned to user.

The following section will demonstrate how Active Server Pages work using real working examples.

a.   ASP File

The first line of the code reveals the scripting language which will be used in the ASP file is VBScript. Writing the name of the scripting language, is a programming practice to to give information for future uses, such as modification of the code.

<% @LANGUAGE=VBScript %>

<% and *%>* scripting delimiters indicate that the scripting code is to be executed on the server.

<% Option Explicit %>

Option Explicit indicates that all VBScript variables should be explicitly declared by the user.

Set up the variables for this page:

<% Dim dbConn, dbQuery, loginRS, loginFound

Check to see if there is an existing connection to the Database. If not, create one:

If IsObject(Session("thesissql_dbConn")) Then

Set dbConn = Session("thesissql_dbConn")

Else

An ActiveX Data Object (ADO) connection *dbConn* is created.

ADO provides a uniform way for a program to connect databases in a generic manner without having to deal with specifics of the database system.

Set dbConn = Server.CreateObject("ADODB.Connection")

Using the newly created ADO connection, the object's open method opens the thesissql system database (DSN).

Call dbConn.Open("thesissql", "", "")

Set Session("thesissql_dbConn") = dbConn

End If

Create the SQL query to return the names of users in database.

*dbQuery = "SELECT * FROM users"*

24

Create the recordset:

Set loginRS = Server.CreateObject("ADODB.Recordset")

Call loginRS.Open(dbQuery, dbConn)

If an error occurs, ignore the error.

On Error Resume Next

Move to the first record in the recordset

Call loginRS.MoveFirst()%>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">

<HTML><HEAD><TITLE>Login Page</TITLE></HEAD><BODY>

<!-- include header goes here-->

<!-- #include virtual="aspSample/includes/mgtheader.inc" -->

ASP file uses a serve side include (SSI) statament to incorparate the contents of *mgtheader.inc*. The SSI statement is basically replaced with the contents of the file *mgtheader.inc*. SSI statements are always executed before any scripting code. The word virtual in in SSI statement refers to a virtual path.

If this is a return after a failed attempt, print an error:

<% If Session("loginFailure") = True Then %>

<FONT SIZE = 4 COLOR = "red"> Login attempt failed,

please try again <P></FONT>

<% End If %>

<% ' Begin the form %>

<FONT FACE = "arial" SIZE = 2>

Please select your name and enter

*your password to login:
*

Create a form whose name is *sublogform,* response action *submitlogin.asp* and method type *POST*.

</FONT><FORM NAME = sublogform ACTION = "submitlogin.asp"

METHOD = POST>

25

```
<% ' Format the form using a table %>

<TABLE BORDER = 0>

<TR><TD>

<FONT FACE = "arial" SIZE = 2>Name:</FONT></TD>

<TD><SELECT NAME = "LOGINID">

<OPTION VALUE = "000">Select your name
```

 Pull user names from the query to populate the dropdown menu.

```
<%    While Not loginRS.EOF
```

   If there is a session login ID, reuse the existing one.

```
    If Session( "loginid" ) = loginRS( "loginid" ) Then

      loginFound = "selected "

    End If
```

   If a login cookie was found, reuse the user ID in the cookie.

```
    If Request.Cookies( "loginid" ) = loginRS( "loginid" ) Then

      loginfound = "selected "

    End If%>
```

Create each dropdown entry:

```
<OPTION <% =loginFound %>

value="<% =loginRS( "loginid" ) %>">

<% =loginRS( "loginid" ) %>

<% loginfound = " " %>

<%   Call loginRS.MoveNext()

    Wend%>

</SELECT></TD></TR>

<TR><TD><FONT FACE="arial"SIZE=2>Password:</FONT>

</TD>

<TD><INPUT TYPE = "password" NAME = "SUBMIT_LOGIN">

</TD></TR>

<TR><TD> 
```

```
</TD>
<TD ALIGN = "LEFT">
<INPUT TYPE = "submit" VALUE = "Log Me In"
    ID = "login1" NAME = "login1">
</TD></TR>
</TABLE></FORM>
</FONT>
<!-- #include virtual="aspSample/includes/mgtfooter.inc" -->
</BODY></HTML>
```
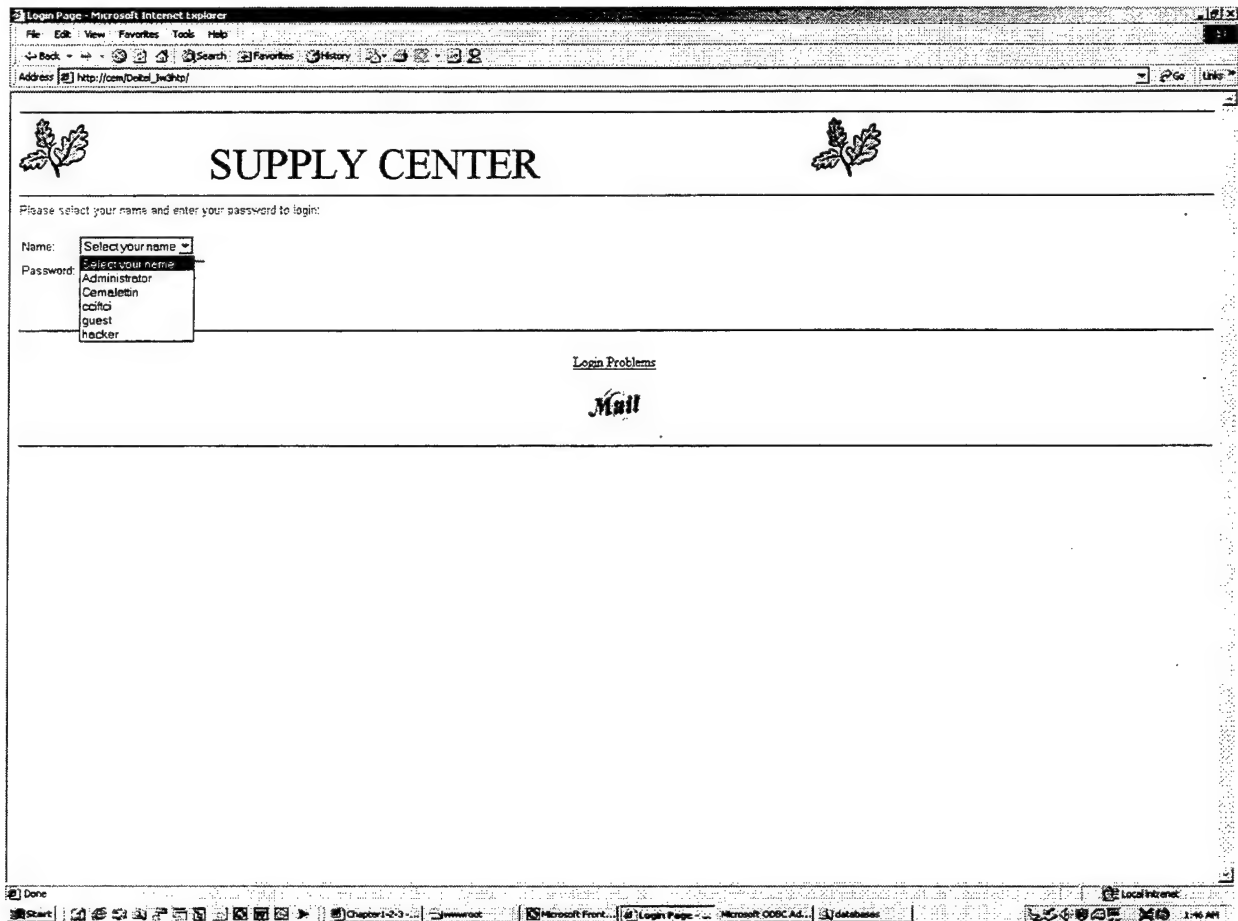

Figure 3.5 ASP Example Screen Snap Shot.

D.   JAVA SERVLETS

A Java servlet is a server-side program that basically services HTTP requests from the clients, and returns results as HTTP responses. This model is called a request-response model, and one of the common implementation of this model is between web browser and web servers.

Servlets can be regarded as non-Gui applets that run on the web server. The execution process of a servlet is also similar to that of applets. Servlets are loaded and executed by a web server in the same manner that applets are loaded and executed by web browsers. If servlets are compared with applets :

Servlets run at the server side.

Servlets are for the server side not for browsing.

Servlets are more secure than applets.

Servlets extend the functionality of the server.

Servlets do not have a visual appreance(faceless) in contrast to applets.

The servlet request – response model can be seen in the following figure:
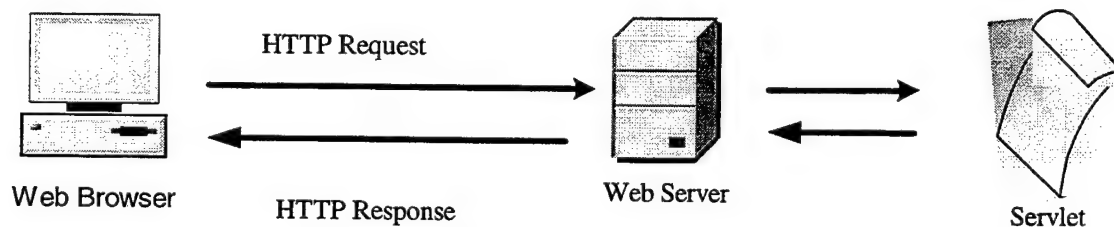
Figure 3.6 Servlet Request – response Model

The basic flow for servlet use can be listed as follows;

1. The client (most likely a web browser) makes a request via http.

2. The web server receives the request and forwards to the servlet

28

3. If the servlet has not yet been loaded, the web server or the servlet engine will load the servlet into a Java Virtual Machine (JVM) and execute it. If the web server does not have built in servlet support Sun Microsystem's servlet engine included in JSDK2.0 or later can be used to add that capability.

4. The servlet will receive the HTTP request, read input parameters, and process them accordingly.

5. The servlet will return a response back to the web server.

6. The web server will forward the response to the client.

7. Servlets are written in Java so the client can benefit from everything that Java has to offer; such as robustness, extensibility and object oriented cvapabilities. Since servlets are an extension of the Java Platform, they provide access to all of the Java APIs. As a result the servlets can send and receive e-mail, invoke Remote Method Invocation (RMI) objects and make use of Enterprise Java Beans (EJB)

1. Requirements for Servlet Use.

Since the Java programming language will be used for writing servlets the Java Development Kit (JDK) version 1.1.1 or later will be required. The Java Development Kit can be downloaded from Sun's web site at no cost.

The Server Development Kit (JSDK) which contains the fundamental classes for servlets will be required as an addition to the basic Java kit. The Server Development Kit also can be downloaded from Sun's web site.

Installing the downloaded kits from Sun, servlets can be written using any programmers editor. However the use of Java IDE's could speed up the learning process for programmers by providing a visual development environment and may save time by using the built in wizard programs. Some of the IDE's supporting servlet development includes Borland's Jbuilder 3.5, Norton's Visual Café, IBM's Visual Age, Sun's Forte.

2.   Servlet Engine (Container)

In order for programmers to save time and to avoid attending to the details of servlet programming the tasks of network connection, request catching and producing responses are performed by a servlet engine, which is also called a servlet container.

In the implementation phase of this thesis, the default servlet engine, which is a part of Java Servlet Development Kit (JSDK) from SUN, will be used.

3.   Servlet Invocation

In order to be able to run servlets as a response to client requests requires that servlets be invoked. The methods for servlet invocation are listed as follow:

- If the servlet is stored in the servlet directory, the servlet can be invoked using the servlet's Unified Resource Locator (URL). An example will look like *http://servername:8080/servlets/servletname.*

Port number 8080 is the default port number on which our servlet engine listenes.

- The Servlet can be invoked in a server-phrased HTML file using the <servlet> tag.

- The web server can be configured to recognize and respond the servlet and automically execute the servlet. The Java Web server is an example of a server that does not require any additional servlet engine.

4.   Life Cycle of a Servlet

If listing the steps of a servlet's life cycle, they would be as follows;

- The servlet engine creates an instance of the servlet.
- The engine calls the servlet instance's *init()* (initialization) method.
- If the engine has a request for the servlet, the engine calls the servlet instance's *service()* method.
- Before destroying the instance, the engine calls its *destroy()* method.
- The servlet instance is destroyed and java garbage collection process occurs.

30

The container may decide at any time to remove the instance from memory if the servlet has not been called within a specific amount of time or the instance may also be removed if the servlet engine is terminating.

5. Motivations For Servlet Use

Because servlets are executing on the server side, the security issues associated with the applets do not apply to the servlets. If the web server is secured behind a firewall, the servlet can be declared secure as well.

Since servlets are written in Java, theoretically, they become platform independent. The program code can be written once and run on any operating system.

Servlets are persistent. As opposed to the CGI scripts, which is a rival programming technology to servlets, servlets are loaded only once when the first request is received. On the other hand, CGI scripts are loaded each time a request is made to the server. So every time a request is made to the CGI scrip, a new process is created, which brings extra load to the system.

Servlets are fast. As servlets are loaded only once, they give better performance compared to other technologies.

E. CONCLUSION

1. Java Servlets vs. ASP

At first glance, Java servlets and ASP technology have similarities, such as the fact that both are designed to create interactive web applications. As discussed in the earlier part of the chapter, both technologies provide the programmer with the option to separate business rules from the whole application, and programming logic from the web page design.

These technologies establish an easier and faster development environment for the programmers by offering an alternative to creating CGI scripts. While these technologies

31

have similarities, there are also a number of differences which are explored in the following section.

a. *Platform and server independence.*

In many ways, the biggest difference between Java servlets and ASP technology lies in the software design. Java servlets like its predecessor Java programming language, is designed to be platform and server independent. In contrast, ASP is a Microsoft technology that basically relies on Microsoft technologies and requires extra software support when working on a different COTS operating system and web server.

b. *Portability*

Java servlet technology uses Java language while ASP uses Microsoft VBScript or Jscript. Java is a mature, powerful, and scalable programming language that provides benefits over scripting languages such as VBScipt. Java provides a component-based model that speeds up application development. Programmers can quickly build complex enterprise applications by using previously developed lightweight components.

c. *Performance*

Since servlets are developed using the Java language they provide superior performance compared to ASP that uses interpreted VBScipt or Jscript languages[Ref.9].

d. *Security*

Java servlets do not cause system crashes. If something goes wrong with the servlet, the servlet engine will terminate the existing servlet without harming the running operating system. On the other hand, it is possible for ASP applications to crash the Windows NT systems. Since servlets use Java, memory management is handled by the Java Virtual Machine (JVM), which protects the system against memory leaks and

pointer errors. Java also provides a detailed exception handling mechanism for real world applications.

e.   *Maintenance*

Applications using Java servlets are easier to maintain than ASP based applications. The mix of script and HTML, basically two sets of information threaded together, can become a maitenance problem for ASP based applications. Scripting languages are fine for small applications but they do not scale well to manage enterprise level applications. Because of the way Java is structured, it is easier to build and maintain large modular applications. Servlet technology emphasizes components over scripting, which makes it easier to reuse content.

2.   CGI Perl vs. Java servlets

As previously stated CGI was the first server side web technology and almost a standard for developing dynamic web pages. The well known language for developing CGI applications is Perl. Servlets, on the other hand are written and compiled using 100 percent Java. This fact is the reason why Servlets are compatible with all operating systems and can be used with any web server.

The following section will compare CGI Perl and Java in terms of structure performance, security, portability, development and error debugging.

a.   *Structure*

Perl uses the concept of modules to extend its capabilities. For example, the Database Independent (DBI) module allows Perl scripts to access databases. These modules are available from Comprehensive Perl Active Network (CPAN) for free.

Java uses packages to increase functionality. For database access, Java Database Connectivity (JDBC), which will be discussed in chapter V, is used. However, a driver for specific database management system is needed in addition to JDBC.

33

b. *Performance*

One of the drawbacks of the CGI environment is the need to create a new process for each request, a requirement that overloads the perl interpreter. The interpreter then loads the script, compiles and executes. If communication with a database is needed, for each CGI request a new connection is established. Because Java is multi-threaded, servlets can be as well. Servlet engines start a new thread for each incoming request. To run the servlet, a java virtual machine and servlet engine is required to run on the server at all times. As a result, Java servlets use much less system resources, scale better and provide improved performance, especially for the busy web sites.

c. *Security*

One of the biggest concerns within the CGI environment is security while processing the confidential user inputs. If the end user somehow tricks the CGI to execute a command on the server, security will be at risk. Compiled languages such as Java provide better security than interpreted scripting languages. Java is not at risk of running unintended shell commands.

d. *Portability*

There are many variants of Perl for almost all COTS operating systems, but the problem is that Perl Unix is not 100 percent compatible with Perl windows NT. It is possible to write CGI programs that can work on both operating systems. However, if the CGI program contains a Unix shell command, the programmers should know that the code will not work on Windows NT platforms. Since its inception Java has been designed to be portable across all platforms. Therefore portability is not an issue for pure Java servlets.

e. *Development*

Perl development is fairly simple and primitive. All that is needed is a text editor and Perl knowledge. There is no visual Integrated Development Environment

(IDE) for perl, that could save time in developing applications. There are many IDE's for Java, that facilitate the work of the programmers. Sharing development among developers is considerably easier with Java. Each programmer can be responsible for developing an individual package, whereas programmers work on CGI applications simultaneously.

3.   Summary

As the needs and policies of the Navy change in time a solution that will provide platform portability and be vendor independent is appropriate for the development and implementation of emerging new systems.

Security will be a concern at all times for the armed forces. To prevent the disclosure of confidential information, dependable technologies should be selected and used during the implementations of enterprise level applications.

Implemetation costs for the system and the Navy's budget restrictions will affect the selections of the decision makers. Therefore a cost effective option will always have priority.

With these stipulations in mind, Java and servlet technology appears to meet the needs of Navy as well as those of the programmers who will be implementing the system. In the implementation section of the thesis, a sample implementation will be developed using Java and servlet technology. Detailed information about servlets and servlets API will also be provided.

35

THIS PAGE INTENTIONALLY LEFT BLANK

IV. DATABASE ACCESS METHODS

In order to be able to communicate with the supply center's database connection software is needed. Whether the connection software comes with database server or is written by the programmers, this software would be essential for database communications and interactions for developing database aware applications.

Although there are innumerable methods of retrieving and storing data from a database server, the following are the most common: Native, ODBC, and SQL. SQL is probably the most common data access method, ODBC a close second, and, except for driver creators, native methods are rarely used [Ref 1]. Figure 4.1 illustrates the software layers in the three methods outlined.
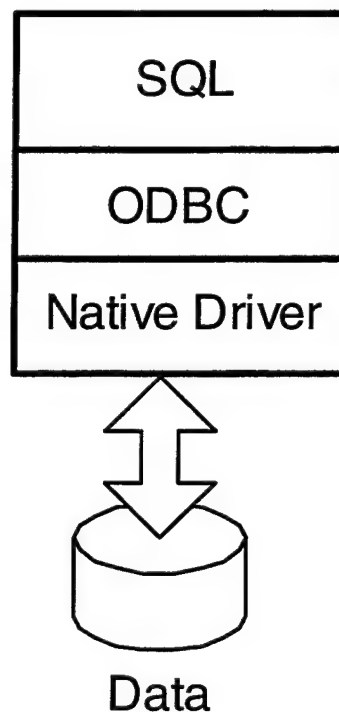
Figure 4.1 Software Layers Used to Access Data

A. NATIVE DRIVERS

DBMS specific drivers or interfaces must be published to allow a client application to communicate with and access the database. The driver is usually in the form of a dynamically linked library (DLL) and resides on the client machine.

Native drivers give the programmer the raw power of talking directly to the database. When the connection is made and data retrieved, the program will be talking directly with the database system. An example of a native driver is the Oracle Call Interface, (OCI) from Oracle Corporation for Oracle databases. Native drivers are usually statically or dynamically linked into the programmer's software at compile time.
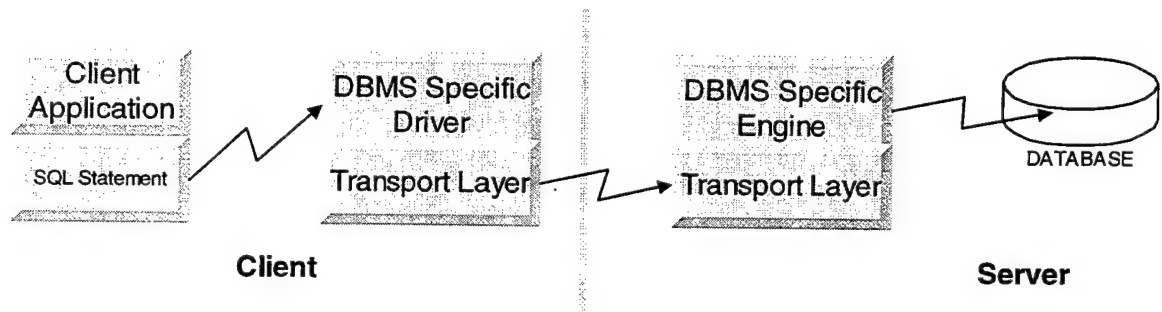
Figure 4.2 Native Driver Architecture

1. Advantages of Native Drivers

The advantages of using native drivers are as follows;

- Since the actual database access code is linked with the running program, data access is very fast.

- In order for programs to access data, no additional mapping or translation is required.

2. Disadvantages of Native Drivers

On the other hand the following disadvantages of using native drivers exist:

- Applications, created using the native drivers, are usually not portable to other platforms without code modifications.

38

- Because the driver is linked into the application, changes in the driver software, such as an update, require possible recompilation of the application.
- Use of native drivers ties the organization to a single vendor solution.
- Since only the features provided by the RDBMS vendor will be avalible there may be limited functionality

B. ODBC

Open Database Connectivity, or ODBC, is a data access standard developed by Microsoft Corporation. ODBC is an application program interface for accessing data in a standard manner regardless of the type and manufacturer of the database. If the data source is ODBC compliant, any odbc client can access it. ODBC drivers are available for almost every major database vendor.
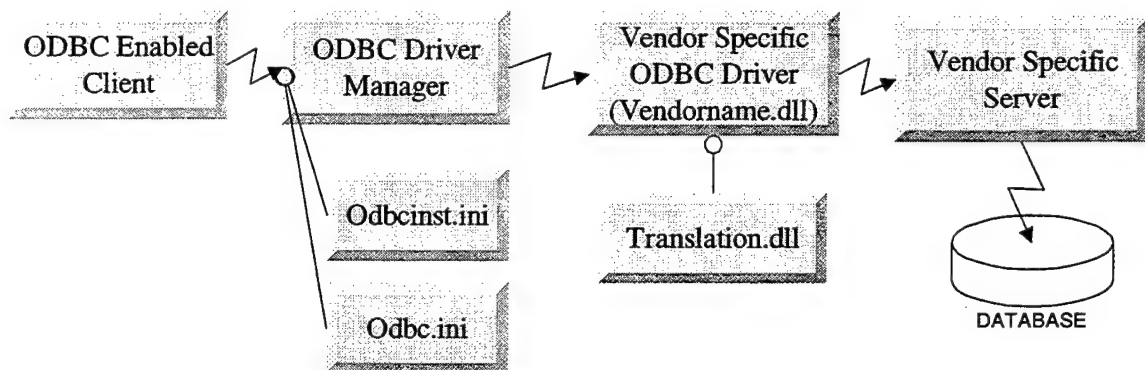
Figure 4.3 ODBC Architecture

1. Advantages of ODBC

The advantages of using ODBC for database access are:

- Since, only one additional layer of software will be used to access the data ODBC is not as fast as native drivers but faster than others.
- SQL can be used to query the database.

39

2. Disadvantages of ODBC

ODBC has the following disadvantages:

- Applications created using ODBC will not be portable to other platforms without code modifications.
- Because the driver stub is linked into the application, changes in the driver software API may require recompilation of code.

C. SQL

Although SQL is not a layer of access to databases like ODBC or native drivers, the Structured Query Language, or SQL, provides a standard method of querying data from different data sources. The following section summarizes SQL commands which will be used in the implementation of this thesis:

1. Commit

Most RDBMS's work with units called *transactions*. A transaction can be made up of multiple actions. The commit command instructs the database to record all the actions which have been performed up until the present time, and to reset the transaction. When commit takes place, the data becomes available to everyone who has access to it. Before the commit occurs, however, only people with access to the database schema can see the changes.

2. Rollback

The rollback command is the opposite of commit. Rollback instructs the database to remove any changes since the last commit. Rollback is very useful for long, multiple-table updates in case an error occurs. For example, suppose that 10 rows need to be added to a table. After inserting 9 rows, the 10th insert fails. The first 9 rows must be removed for the data to retain its integrity. Using the rollback command, the 9 inserted rows will not be recorded.

3. The Where Clause

Most SQL commands act on all the rows of a table at one time. These global actions can be restricted to a limited number of rows by the use of a *where clause*. The where clause allows programmers to specify criteria which is used to limit the number of rows on which an action is performed.

The general syntax for a where clause is as follows:

COMMAND arguments WHERE [[[schema.]table.]column OPERATOR value]

[ANDIOR[[[schema.]table.]column OPERATOR value]]

where

arguments: are the arguments specific to the COMMAND.

Schema: is the area where the table exists

Table: is the table where the column lives

Column: is the column name to compare with *value* parameter.

Value: is a literal or column name to compare with column

Multiple operations may be checked in the WHERE clause. These operations can be linked with either the AND or OR keyword.

The OPERATOR might be many things depending on the RDBMS in use. Table 4.1 shows the OPERATORs that are available in most RDBMSs.

Operator	Meaning	Example
<	Less than	Salary <100
>	Greater than	Salary >2000
=	Equal to	Selection = "Y"
<=	Less than or equal to	Salary <= 1500
>=	Greater than or equal to	Salary >= 1750
<>	Not equal to	Age <> 25
is	For checking NULL values	Age is NULL
not	For negating an operator	Age is not NULL
like	Allows for the use of wildcards	Name like %Cem%

Table 4.1 SQL Operators

The WHERE clause cannot be used alone, it must be appended to a DELETE, SELECT or UPDATE command.

4.   Insert

The insert statement allows programmers to create new rows in a table.

The syntax for an insert statement is as follows:

INSERT INTO[schema.]table.[(column [,column…..])]VALUES (value [,value])

where

schema: is where the table exists

Table: is the target table

Column: is the column names of the data to insert.

value: are the values to be inserted into table.

Examples

INSERT INTO REQUEST (INSTID, NIIN, REQUESTTIME, QUANTITY, PRIORITY, STATUS) values (1, 000793231, 13, 3, 01/20/2001, NEW ENTRY)

5.   Delete

The DELETE statement allows the removal of a row or rows from a table.

The syntax for a DELETE statement is as follows:

DELETE FROM [schema.]table [WHERE expression]

where

schema: is where the table exists

Table: is the target table

Expression is an expression as outlined in the preceding WHERE clause section

Note that without a WHERE clause, DELETE command removes all rows from a table.

Examples

DELETE FROM REQUEST WHERE INSTID =1

6.   Select

The SELECT statement allows the programmer to retrieve a row or rows from a table.

The syntax for a SELECT statement is as follows:

SELECT   [[schema.]table.]column   [,[[schema.]table.]column]   FROM [schema.]table [WHERE expression]

where

schema: is where the table exists.

Table: is the target table.

Column: is column or columns to retrieve.

43

Asterisk ('*') can be used to indicate that the SELECT statement should return all columns.

Expression is an expression as outlined in the preceding WHERE clause section.

Examples

SELECT QUANTITY FROM REQUEST WHERE NIIN = 000793231

7. Update

The UPDATE statement allows the programmer to modify a column or columns in one or more rows in a table.

The syntax for an UPDATE statement is as follows:

UPDATE   [schema.]table   SET   [[schema.]table.]column   =   value [,[[schema.]table.]column = value][WHERE expression]

where

schema: is where the table exists

table: is the target table

column: is column or columns to modify

expression: is an expression as outlined in the preceding WHERE clause section

value: is the new value that the column should hold

Examples

UPDATE REQUEST SET STATUS = PROCESSING

8. Advantages of SQL

- The layman does not need to know how to program to use SQL, because simple English syntax is used.
- SQL uses simple English words to instruct the database to perform certain actions. SQL can be used with almost every major database product available today. In addition, programmers can even use SQL syntax to interact with a data source using ODBC.
- Standardized by American National Standards Institute (ANSI).

44

9.   Disadvantages of SQL

- SQL queries can become quite complex and lengthy. Because SQL syntax uses standard English words, the resulting SQL statements can get somewhat monotonous for the programmers.

THIS PAGE INTENTIONALLY LEFT BLANK

V. JAVA DATABASE CONNECTIVITY (JDBC)

In an effort to create an independent database standard API for Java, Sun Micro Systems developed Java Database Connectivity, also known as JDBC. JDBC offers a generic SQL database access mechanism that provides a consistent interface to a variety of RDBMSs. JDBC's ability to establish a consistent interface is achieved through the use of database driver modules. Database vendor wishing to have JDBC support for their product, must provide the driver for the platform on which the database and Java run.

To gain wider acceptance of JDBC by the programmers, Sun based JDBC's framework on ODBC. As discussed in the previous chapter, ODBC has widespread support on a variety of platforms. Basing JDBC on ODBC allows vendors to bring JDBC drivers to market much faster than developing a completely new connectivity solution.

A. JDBC GOALS

The creators of JDBC have designed this software package with a number of specific goals and objectives which drove the development of the API. As a result of JDBC's success in achieving these objectives, the JDBC class library has established itself as a solid framework for building database applications in Java[Ref15].

The goals which have been set for JDBC will be studied in this chapter in order to give some insight as to why certain classes and functionalities behave the way they do. The seven design goals for JDBC can be listed as follows:

SQL Level API

SQL level API means that JDBC allows the programmers to construct SQL statements and embed them inside Java API calls. In short, programmers use SQL and JDBC will translate between database and Java applications.

47

SQL Conformance

SQL syntax may vary depending on the database vendors who are aiming to extend the capabilities of their product. In an effort to support a wide variety of vendors, JDBC will allow any query statement to be passed through to the underlying database driver. This feature allows the connectivity module to handle non-standard functionality in a manner that is suitable for users.

Implementable On Top Of Common Database Interfaces

The JDBC SQL API must be placed on top of other common SQL level APIs. This goal allows JDBC to use existing ODBC level drivers by the use of a software interface. This interface would translate JDBC calls to ODBC, and vice versa.

Consistent interface with Java

Because of Java's acceptance in the user community, the designers did not stray from the current design of the core Java system.

Simple

Software designers try to keep their product as simple as possible to gain more acceptance through the programmaers community, and JDBC is no exception. Sun felt that the design of JDBC should be very simple, allowing for only one method of completing a task per mechanism. Allowing duplicate functionality only serves to confuse the users of the API[Ref16].

Common cases

Because typical SQL calls used by the programmer are simple SELECT's, INSERT's, DELETE's and UPDATE's, these queries should be simple to perform with JDBC. However, more complex SQL statements should also be possible for advanced user.

Use of multiple methods to express multiple functionality

There are guiding principles that determine functionality. One is to provide a single entry point into an API. The programmer must then use a variety of control parameters to achieve the desired result. The second is to provide multiple points of entry into the API. The latter is the guiding principle for JDBC.

B.   JDBC OVERVIEW

JDBC is basically divided into two parts: the JDBC API, and the JDBC Driver API. The JDBC API is the programmer's API. This half is where the programmers spend time coding. The JDBC Driver API is designed for driver writers and database vendors to create connectivity modules for their database software. The figure below shows the complete JDBC API class hierarchy.
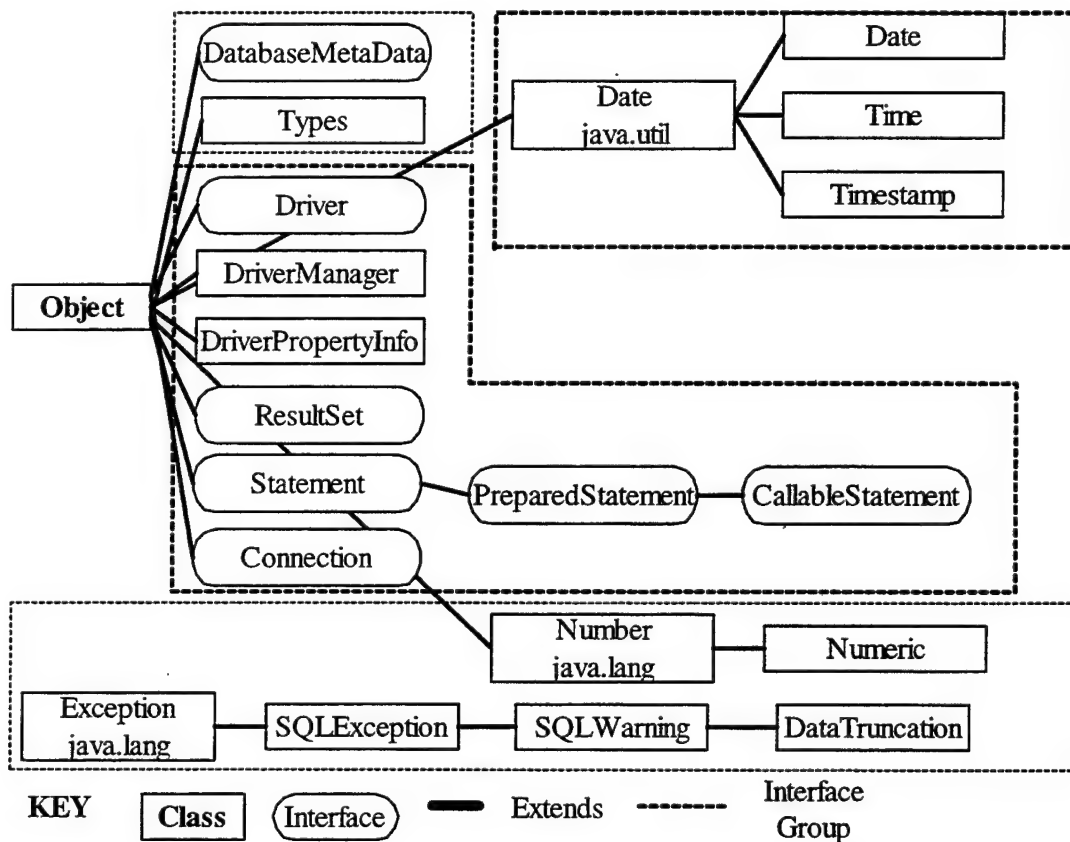
Figure 5.1 JDBC Class Hierarchy

49

The JDBC API consists of many classes and interfaces. This structure makes the API a semi-abstract set of functionality. In order for JDBC to be of any use, a database vendor must provide related classes.

Orfali and Harvey defines the DriverManager interface as the backbone of JDBC whose main purpose is to connect Java applications to JDBC drivers. As seen in the following figure JDBC-ODBC bridge enables JDBC to use ODBC to access ODBC aware databases such as Microsoft SQL Server 7.0 . JDBC-ODBC bridge can be used to access ODBC compliant databases for which a specific JDBC driver is not provided[Ref18].
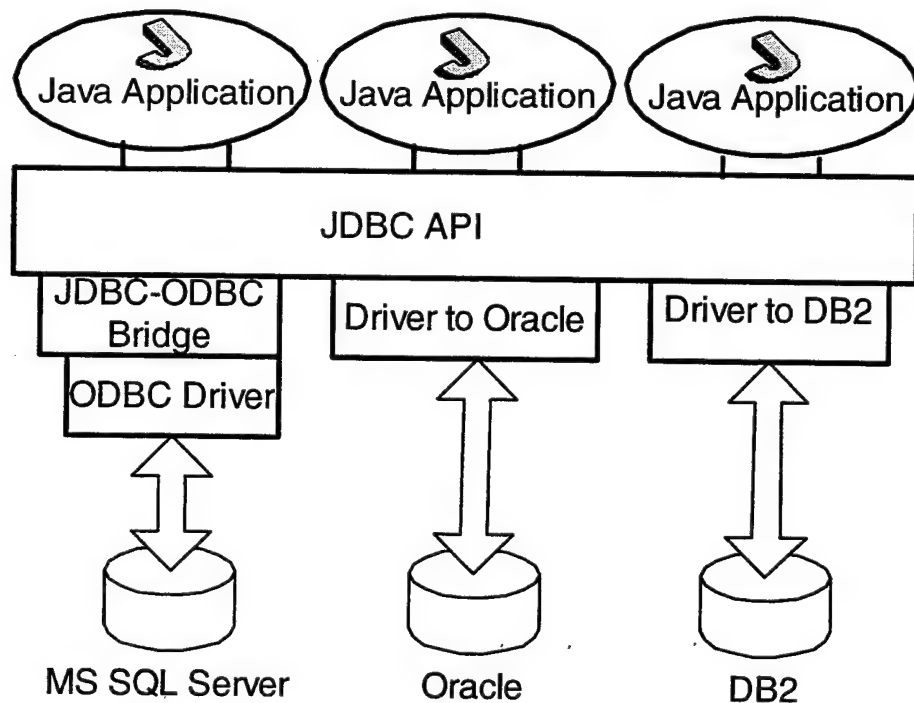
Figure 5.2 The Layers of JDBC

50

1. JDBC Interfaces

JDBC interfaces are classified in the following four groups

a. *JDBC Core Interfaces*

These are the main interfaces and abstract classes that should be implemented by every JDBC driver. Four of these classes are the focus of any database programming, which perform almost 90 percent of any database application. These four class and interfaces can be seen in the figure and listed as follows:
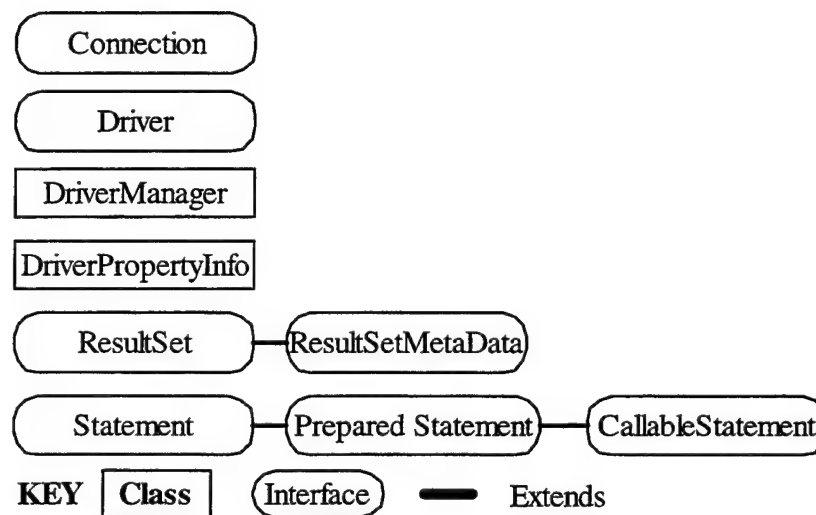
Figure 5.3 Core Classes and Interfaces

(1) DriverManager. This class is responsible for managing all the available database drivers. The DriverManager class retrieves the list of available classes for drivers from the system property called *jdbc.drivers*. Each of the located drivers is loaded.

(2) Connection. This interface defines a session between an application and a database.

(3) Statement. This interface is used to issue a single SQL statement through a connection. Each statement owns only one ResultSet. Therefore, multiple concurrent SQL statements must be done through multiple statements. Issuing a SQL statement while processing the ResultSet of a previous statement will result in the overwriting of the previous results.

(4)   ResultSet. This interface provides access to the data returned from the execution of a Statement.

b.   *Java Language Extensions*

These extension classes provide JDBC high-precision data types and its own exception and warning types as seen in the following figure.
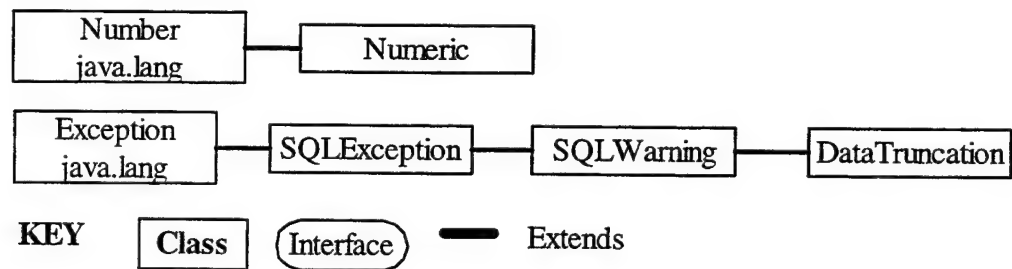
Figure 5.4 Java Language Extensions

c.   *Java Utility Extensions*

Utility extensions provide fine grained time and date utilities, which allows measurement of the time in nanoseconds. These classes are shown in the following figure.
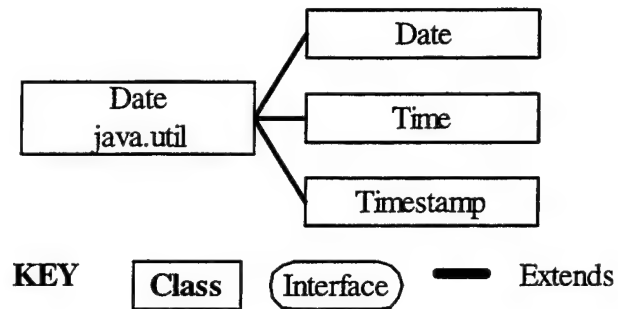
Figure 5.5 Java Utility Extensions

d. SQL MetaData Interface

This interface standardizes access to database metadata across multiple DBMS vendors. This interface consist of 133 methods that should be implemented by JDBC vendors[Ref19].
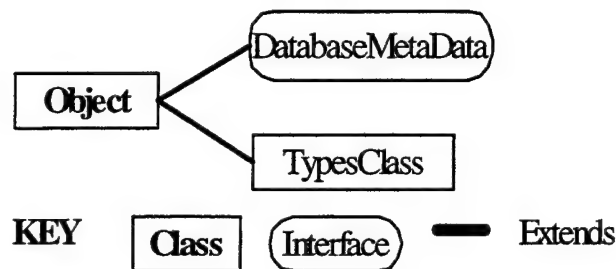
Figure 5.6 java.sql MetaData Interface and The Types Class

This short overview of JDBC is only a small portion of the JDBC API. JDBC also supports other advanced database features such as cursors and stored procedures which are out of the scope of this thesis.

C.   JDBC DATABASE CONNECTION

Since JDBC will be used for database access in the application program for the thesis, the steps in accessing the database and the interfaces used during the process will be explained here.

Basically, the steps to be taken to access a database are the following:

- Loading the database specific driver

- Creating a connection to the database

- Submitting SQL statements

- Retrieving and processing the query results.
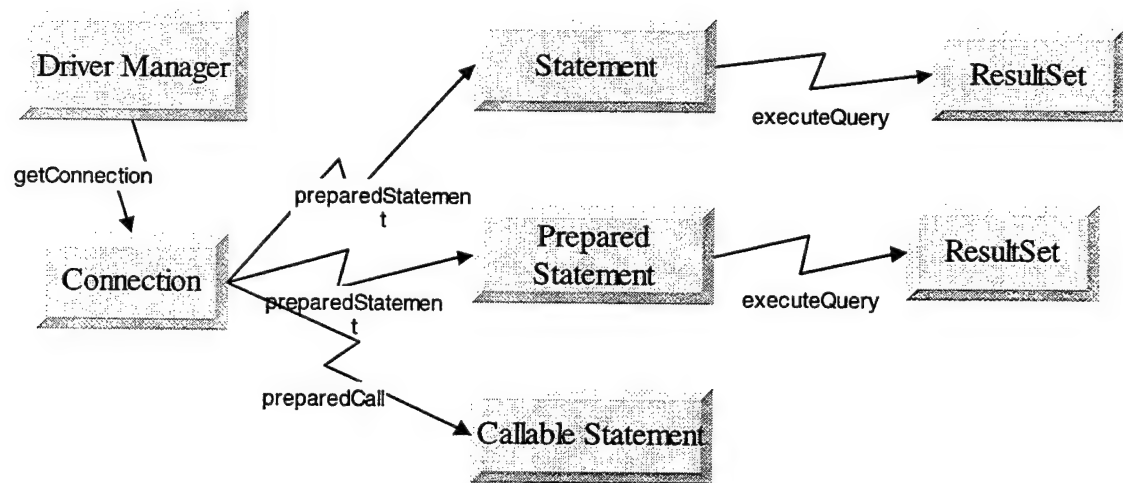
Figure 5.7 Database Access Steps and Interfaces

1.   Loading the Driver

The database drivers can be thought of as bridging technologies to access databases. Commercial JDBC drivers can be found for almost all database management

54

systems. Since cost effectiveness and system independent portability are being sought out by the decision makers for the Navy, JDBC-ODBC bridge will be used .

The driver is loaded by asking for an instance from *Class.forname* which creates and registers an instance automatically, as in the following example.

Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

2.   Creating a Connection to Database

Database URL, userID, password and any properties required by the JDBC driver will be used to request a connection from the DriverManager. After the DriverManager searches through all known *java.sql* driver implmentations for the one that is requested by the program with URL provided and found one, a connection will be created and returned.

String user = "Administrator";

String password = "seval1975";

String url = "jdbc:odbc:THESISSQL";

Connection con = DriverManager.getConnection(url,user,password)

3.   Submitting SQL Statements

The statement interface is a mechanism to execute SQL statements over a database connection. A statement can be obtained by using the *createStatement()* method of the connection object. The SQL statement to be executed is supplied to one of the appropriate execute methods of the statement object which are listed below.

- executeQuery() used for SELECT queries and returns a resultset.
- executeUpdate(), used for UPDATE queries, returns an integer number of rows affected .
- exeute() is used for either queries or updates, or when the statement will return more than one result set.

*String query = "SELECT * from Request";*

Statement stmt= con.createSatetement();

55

stmt.executeQuery(query);

4.   Getting The Query Result

Depending on the statement executed, the results of the query are displayed in appropriate manner. For an executed SELECT statement, one which is mostly used to retrieve information from an enterprise database, a *ResultSet* object will be returned after the execution of the query. By using the *next()* method of the *ResultSet* object, tuples in the set can be accessesed one by one in the order they are received.

*String query = "SELECT * from Request";*

Statement stmt= con.createSatetement();

ResultSet rs = stmt.executeQuery(query);

while(rs.next()){ String status = rs.getString("status");

D.   CONCLUSION

.

Java Database Connectivity is a competing database access method that provides the programmers with an easy to use interface. And its simplicity attracts programmers.

JDBC is a part of Java Development Kit and includes the free driver JDBC-ODBC which will be used as a bridge to the supply center's databases. In the application program for this thesis Microsoft SQL Server 7.0 will be used as the database management system and JDBC will provide access to the database.

56

VI.   IMPLEMENTATION

A.   SERVLET API OVERVIEW

Since Java servlets are implemented in the prototype program, the servlet API will be reviewed.

The servlet API is organized into two-packages, javax.servlet and javax.servlet.http The first package is more general and assumes a basic client-server framework with a stateless request-response model. The second package is specific to HTTP and understands the HTTP protocol and related issues such as HTTP methods and headers. The javax.servlet package contains a generic servlet class that implements a service() method to process a request-response cycle. The javax.servlet.http package has an HttpServlet class that implements a number of doxxx() methods to process a request-response cycle, where xxx corresponds to various HTTP methods, the two most common are doGet() and doPost().

There are four interfaces, which are frequently used by programmers, in the javax.Servlet package. These interfaces are: Servlet, ServletConfig, ServletRequest, ServletResponse and are depicted in the following figure:

Servlet is the basic interface that declares the life-cycle methods of the servlet, which are init(), service() and destroy(). ServletConfig declares methods needed for initialization; the init () method takes one argument, a ServletConfig object, supplied by the servlet engine.
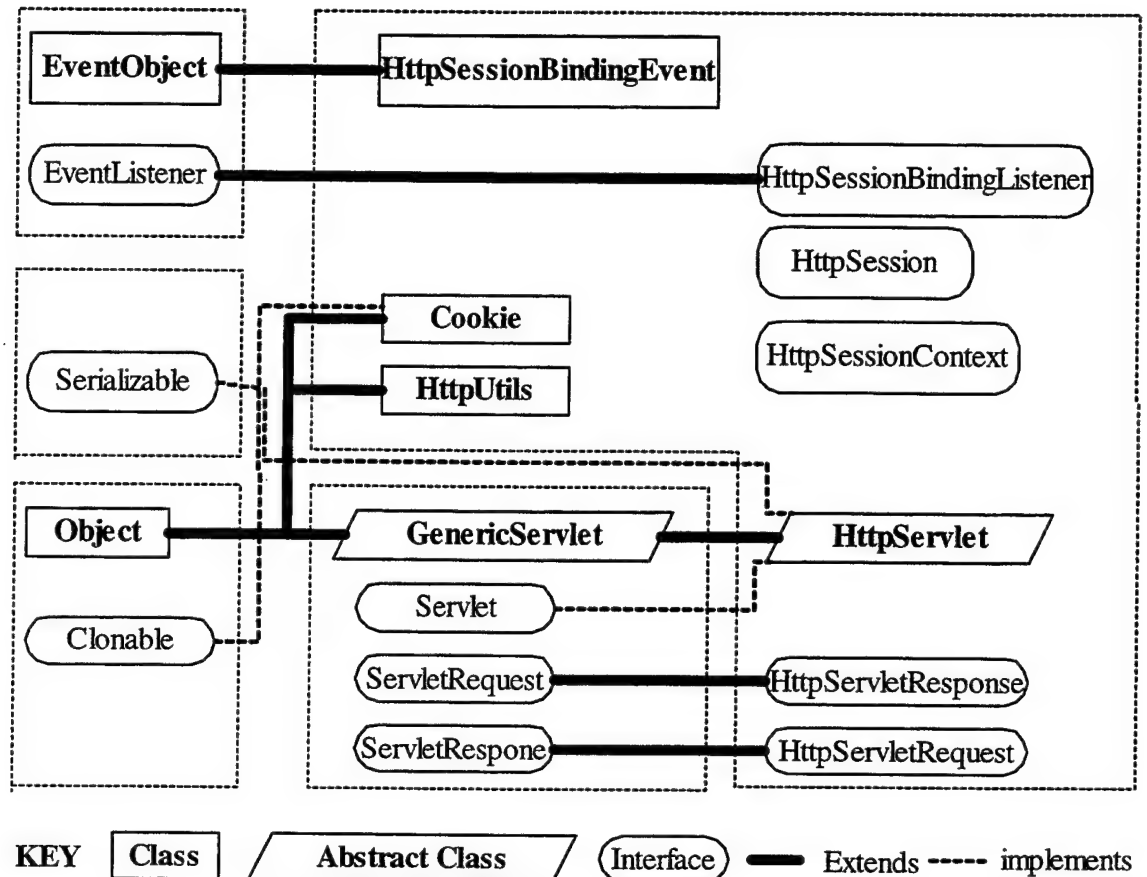
Figure 6.1 javax.servlet.http package API from: [Ref 6]

ServletRequest and ServletResponse declare all the methods needed by the service() method. The method takes two arguments, a request and a response pair that implement the two interfaces. The arguments are supplied by the servlet engine.

1.   Loading and Instantiation

As discussed in the Chapter 3, servlets are Java classes whose role is to connect a web server (or several servers including database servers) to other computational resources and entities. A servlet is not a stand alone program; servlets are invoked by a server in response to specific requests. When a request arrives, the servlet engine makes use of a custom class loader. The class loader checks to see whether the servlet class is already loaded and if the loaded version has the same time stamp as the disk version. If

58

the servlet class is not loaded or is older than the disk file, the engine loads the class into the special JVM provided for that purpose and creates an instance of the servlet. A detailed diagram for servlet loading and instantiating is shown in the following figure[Ref.11].
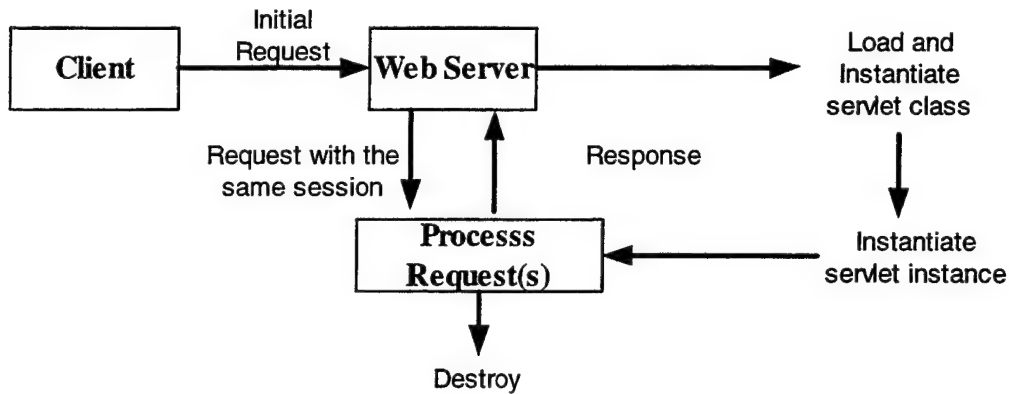
Figure 6.2 Servlet Loading and Initializing Process

2.   Session Tracking

There are a number of problems arising from the fact that HTTP is a "stateless" protocol. For example, when web users are shopping on-line, a web server can not easily remember previous transactions. This problem makes applications like shopping carts very problematic: when users add an entry to their cart, how does the server know what's already in the cart?

There are three standard solutions to the problem discussed above.

a.   Cookies

HTTP cookies can be used to store information about a shopping session, and each subsequent connection can look up the current session and then extract any information about that session from the server machine. Cookies are the most widely-used approach to session tracking on the Internet. However, even though servlets have a higher-level and easier-to-use interface than cookies, there are still a number of details that need to be addressed which will be discussed below[Ref6]:

- Extracting the cookie that stores the session identifier from other cookies (there may be many cookies),

- Setting an appropriate expiration time for the cookie (sessions interrupted for 24 hours or longer may need to be reset); and,

- Associating information on the server with the session identifier (there may be far too much information to actually store in the cookie, plus sensitive data like credit card numbers should never go in cookies).

b.   URL Rewriting.

It is possible to append extra data at the end of each URL which identifies the session, and the server can associate that session identifier with data it has stored about that session. URL rewriting is a good solution to session tracking, and has the advantage that it works with browsers that do not support cookies or where the users have disabled cookies, for their browsers. However, URL rewriting has most of the same problems as cookies, such as it requires extra processing by the server-side program. In addition, programmers need to ensure that every URL returned to the user (Location fields in server redirects) has the additional information appended. Also, if the user leaves the session and comes back via a bookmark or link, the session information can be lost. Finally URL rewriting cannot be used with static HTML pages (all pages must be dynamically created), due to the fact that the URL must be encoded for the user to include a session ID [Ref 7].

c.   Hidden Form Fields

HTML forms can have an entry that looks like the following:

<INPUT TYPE="HIDDEN" NAME="session" VALUE=".......”

The previous statement means that, when the form is submitted by the user, the specified name and value are submitted as GET or POST data. Hidden form fields can be used to store information about the user's session. However, it has a

disadvantage that it only works if every page is dynamically generated, since each session has a unique identifier.

The Java servlet API provides programmers a mechanism for tracking session data. which makes use of two different objects, "cookies" and HttpSession [Ref6].

An HttpSession object is used to store session data in the current servlet context whereas cookies are used to match a particular user with their associated session object by using a session ID. Cookies makes it possible to associate each user with a large amount of data on the server by sending a small amount of data to the client. However in the case of browsers which do not allow cookies, URL rewriting can be used to pass session IDs between requests.

Servlets provide a good technical solution to session tracking, the HttpSession API. HttpSession is a high-level interface built on top of cookies or URL-rewriting. In fact, on many servers, programmers use cookies (if the browser supports cookies) but automatically revert to URL-rewriting when cookies are unsupported or explicitly disabled. The HttpSession API provides a convenient place to store data that is associated with each session so the programmers do not have to explicitly manipulate cookies or information appended to the URL.

d. *The Java Servlets Session Tracking API*

Using sessions in servlets is very straightforward, and involves looking up the session object associated with the current request, creating a new session object when necessary, looking up information associated with a session, storing information in a session, and discarding completed or abandoned sessions. Each step will be discussed to illustrate session tracking using the Servlet API.

(1) Looking up the HttpSession object associated with the current request. This is done by calling the *getSession* method of HttpServletRequest. If *getSession* returns null, a new session can be created, but this is so commonly done that

61

there is an option to automatically create a new session if there is not one already. The first step looks like following:

HttpSession session = request.getSession(true);

(2) Looking up Information Associated with a Session. HttpSession objects live on the server; these objects are automatically associated with the requester by a mechanism (e.g.cookies or URL-rewriting). These session objects have a built-in data structure that allows programmers to store any number of keys and associated values.

In version 2.1 and earlier of the servlet API, *getValue("key")* can be used to look up a previously stored value. The return type is Object, so the programmers must do a typecast to a more specific type of data which has been associated with the key in the session. The return value is null if there is no such attribute. In version 2.2, *getValue* is deprecated in favor of *getAttribute*, because of the better naming match with *setAttribute* (the match for *getValue* is *putValue*, not *setValue*), and *setAttribute* allows programmers to use an attached *HttpSessionBindingListener* to monitor values, while *putValue* does not.

Since Java Development Kit (JDK) version 1.2.2 is being used for development purposes, Servlet Development Kit (JSDK) version 2.0 will be used for compatibility issues. Below is an example, assuming ShoppingCart is a class which is defined by the programmer that stores information on items being purchased.

HttpSession session = request.getSession(true);

ShoppingCart previousItems = (ShoppingCart) session.getValue ("previousItems");

if (previousItems != null) {

doSomethingWith(previousItems); }

else { previousItems = new ShoppingCart(...);

doSomethingElseWith(previousItems); }

In most cases, programmers use a specific attribute name, and wish to find the value (if any) already associated with the session. However, it is also possible to discover all the attribute names in a given session by calling *getValueNames,*

which returns a string array. In version 2.2, *getAttributeNames* is used, which better suits the naming convetion and which is more consistent in that it returns an Enumeration object, similar to the *getHeaders* and *getParameterNames* methods of *HttpServletRequest.*

Although the data that was explicitly associated with a session is most critical for programmers, there are other pieces of information which are sometimes useful as well.

(a) getId.

This method returns the unique identifier generated for each session. It is sometimes used as the key name when there is only a single value associated with a session, or when logging information about previous sessions.

(b) isNew.

This method returns true if the client (browser) has never seen the session, usually because it was just created rather than being referenced by an incoming client request. It returns false for preexisting sessions.

(c) getCreationTime.

This method returns the time in milliseconds at which the session is initiated. To obtain a value useful for printing out the time, the value should be passed to the *Date* constructor or the *setTimeInMillis* method of *GregorianCalendar*.

(d) getLastAccessedTime.

This method returns the time in milliseconds at which the session was last sent from the client.

(e) getMaxInactiveInterval.

This method returns the amount of time in seconds that a session should go without access before being automatically invalidated. A negative value indicates that the session should never time out.

(3) Associating Information with a Session. As discussed in the previous section, information is associated with a session by using the *getValue* (or *getAttribute* in version 2.2 of the servlet API). To specify information, *putValue* is used (or *setAttribute* in version 2.2), supplying a key and a value. The *putValue* method replaces any previous values associated with a duplicate key. Here is an example:

HttpSession session = request.getSession(true);

session.putValue("referringPage", request.getHeader("Referer"));

ShoppingCart previousItems =   (ShoppingCart)session.getValue ("previousItems");

if (previousItems == null) {

previousItems = new ShoppingCart(...); }

String itemID = request.getParameter("itemID");

previousItems.addEntry(Catalog.getEntry(itemID));

session.putValue("previousItems", previousItems);

Since the cart may be new, (and thus not already stored in the session), *putValue* is needed in addition to modifying the cart.

3.   Connection Pooling With Java Servlets

a.   *Problem*

Many database systems limit the number of connections open to a specific database at a given time. For this reason, a connection cannot be dedicated to one user. If the connection is assigned to one user, it will be open for a user during an entire session, even though only a small percentage of the session lifetime will be spent running queries [Ref10].

As a quick solution to the problem listed above, a new connection can be opened prior to each query and closed immediately, afterwards.This adds a significant amount of overhead by repeatedly opening and closing a connection.

b.   Solution

A pool of connections, that share access to the same database with the same driver and list of queries, will be created. When a user wants to run a query, the manager of the connections (connection manager) will obtain a connection from the pool and give it to the user in order for the user to execute the query. When the query has been executed by the user, the connection is returned to the pool [Ref.13].

When deciding to implement a connection pool, database security and thread-safety should be encompassed in the solution.

(1) Database Security. If different users with different access privileges use connections from the same pool, there will probably be a security problem. One solution to this problem could be to create pools for each level of privilege. Even so if a user obtains a connection recently used by someone else at his level, he could access the private data of the previous user. As an improved solution, queries will be constructed for each individual *HttpSession* and stored in session object [Ref.11].

For the sake of simplicity and accuracy, a solution to the security problem in the application program is not provided. It is assumed that everyone who accesses the program has the same level of privilege, username and password couple.

(2) Thread-Safety. Connections in the pools are shared by user sessions, each running in its own individual thread. Unless precautions are taken, connections would not be thread safe for security reasons. Therefore the connection manager should be implemented in such a way that only one instance should exist at a given time. Second, all methods accessing the connections should be declared, "synchronized" [Ref.11].

65

(3) Connection Pool and Caching. An area of memory used to store computationally expensive resources for fast access is called cache. So we can say that caching and connection pooling go together. The implementation of a ConnectionManager will be an instance of a cache.
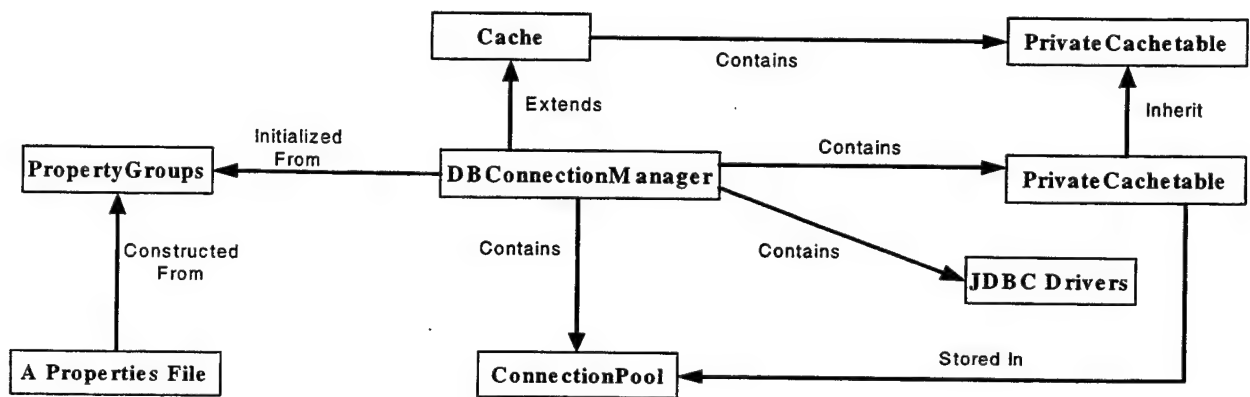
Figure 6.3 Recommended Connection Pooling Architecture

B. DESIGN AND IMPLEMENTATION SMART SYSTEM

1. Design

A logical plan should be followed in the design process of a distributed three-tier system. The first phase is to identify the objects which will be used in the system. The second step of the design phase is to produce the objects in the system and define the interaction between the objects. The Classes with specific responsibilities are identified in this phase. Figure 6.4 shows the simplified class diagram of application program.
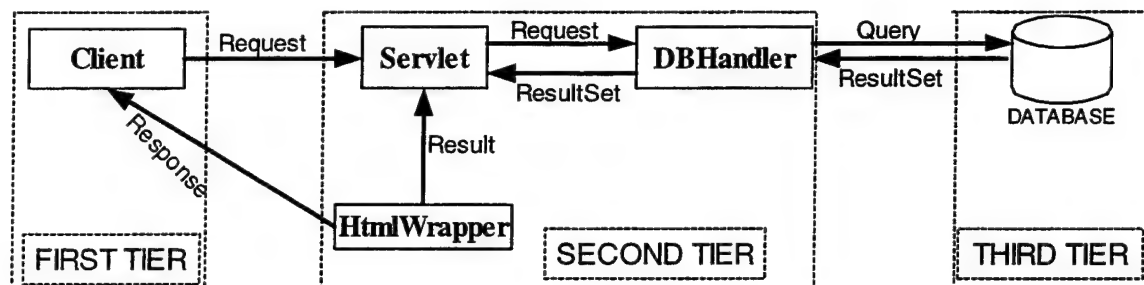
Figure 6.4 Implementation Class Diagram

In the design phase, a sample database for supply centers is created using Microsoft SQL Server 7.0 and populated with data for future evaluations. By using the data subjects, attributes, and relationships identified in the following table, a data model for the supply centers was developed using the Semantic Object Modeling (SOM) technique to provide a better conceptual understanding of the requirements.

The SOM data modeling technique, is similar to the Entity-Relationship (E-R) technique in that both of these techniques facilitate the understanding and documenting of the user's data. The principle difference between these models is that E-R modeling considers the entities and their relationships the atoms of the data model whereas SOM takes the concept of semantic object as basic. These semantic objects are a map of the objects that the users consider important. The semantic objects are the smallest distinguishable units the user may want to process. These objects may be decomposed later into smaller parts in the DBMS, but during the modeling process these smaller parts are of no interest to the user [Ref.19]. The SMART data model consist of following semantic objects: installation, item, request and supplier.

2. Proposed Relational Model

The first step in the development of the Supply Centers Material Request and Tracking System (SMART) database model is to discover the candidate entities. These entities, at the enterprise level, are referred as data subjects. The data subjects for SMART are listed in the following table

67

Data Subject Name	Data Subject Definition
Item	Contains general information about all the items in the inventory of the supply centers such as item name
Installation	All the military installations such as bases, ships and etc.
Orders	An order is any request for an item by the installations
Management information	This data subject contains the management information for an item
Characteristics Information	This data subject contains the characteristics of an item
Reference information	This data subject contains the reference information for an item
Supplier	Information related to the supplier of an item is contained by this object.

Table 6.1 List of Data Subject

The semantic data model should be self-explanatory. Since the data model is kept simple for the sake of the application development, the information given should be sufficient for the reader to understand the supply center database structure.

3.   Application Program Implementation

The purpose of this application is to allow authorized Navy personnel to submit material requests without using standard postal systems. The architecture of the SMART application program is shown below.
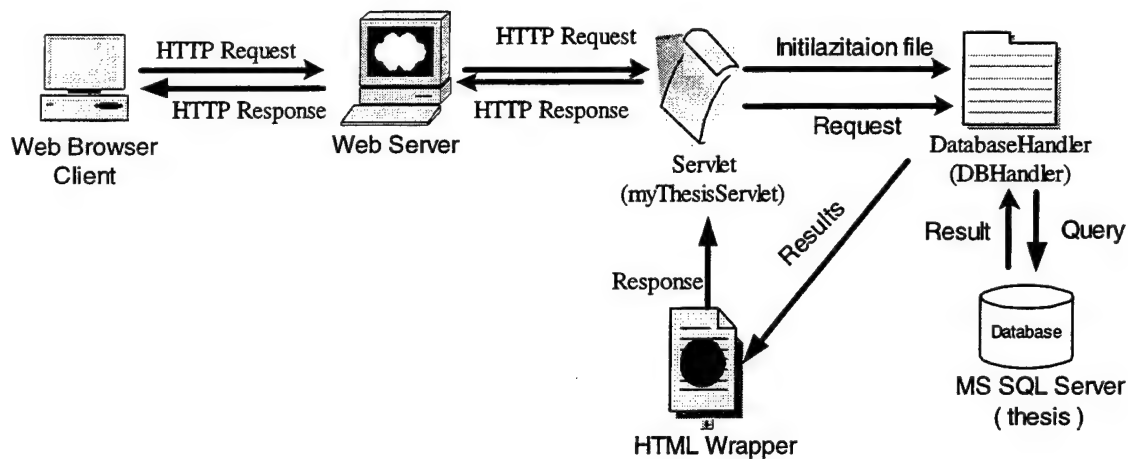
Figure 6.5 Application Program Architecture

The application program was developed using the Java programming language, Java Servlets Application Programming Interface and Java Database Connectivity, that allows the Java servlets to communicate with database server (Microsoft SQL 7.0) using the structured Query Language (SQL) commands. JDBC provides an object-oriented application program with the ability to communicate with Microsoft SQL 7.0 Relational Database Management System via a JDBC-ODBC bridge.

The application program consists of a graphical user interface (GUI), which is created with HTML and control logic created with Java servlets, and Java allowing users to access the supply centers' data stored in Microsoft SQL 7.0 relational DBMS. The program provides users a search menu, input form and update from which will be further discussed in the following section.

69

a.   *Main menu*

The main menu allows the users to enter their user ID and password to access the program and select sub menus. The main menu lists the names of the sub menus in a dropdown list. The default submenu selected is "Stock Number Search Menu". After users logged on using their ID and password couple, they are transferred to the related sub-menu page, depending on their selection.
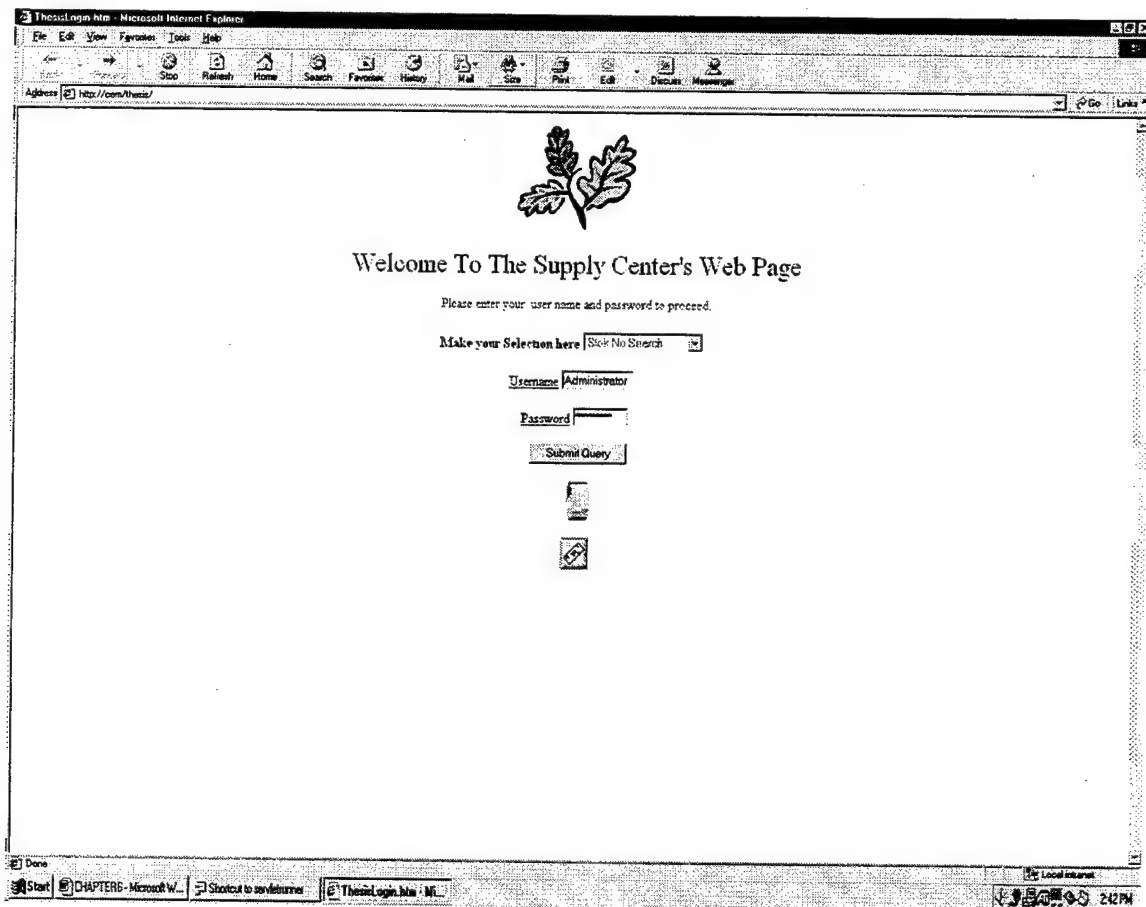
Figure 6.6 Main Menu

b. Stock Number Search Menu

The Stock Number Search Menu enables users to find the Stock Number of items they wish to request by using reference numbers found on the item.

Users can obtain additional information, such as management information, characteristic properties by using the related options listed in the menu.

Figure 6.7 Stock Number Search Menu

c. **Request Enter Form**

A request input form is used to enter requests for further processing by the supply center. This is the primary interface for users to submit their requests. Users are expected to provide their installation ID, the stock number of the item, request quantity and request priority.

Figure 6.8 Request Enter Form

d. Request Check/Modify Form

Users can check the status of their requests by using the provided query menu. It is also possible to make corrections or modifications to any requests which has already been entered into the database.

Figure 6.9 Request Check/Modify Form

73

C. IMPLEMENTATION CLASSES

To promote better understanding of the application program and related java classes, each class will be discussed briefly in the following section. The purpose and the roles of the classes in the application program will be stated and related samples will be provided as necessary.

1. ThesisServlet

This is the main servlet, and basically consists of six parts

- Create a logger, an HtmlWrapper object and session object.
- Create an environment object from the request object and add the URL of the servlet to the environment.
- Obtain username password and create DBHandler for session.
- Depending on the value of the dbOperation parameter (which is hidden in the Html pages submitted by the user), if it is null display the control page.
- If dbOperation is Logout invalidate the session.
- If dbOperation is a query name, which is defined in the startup file, then execute the requested query.

2. Logger Class

Even though the Servlet API provides a log () method, for administrative and debugging purposes it is good idea to use our own logger class. By implementing our own logger class we will be able to:

- Direct the logs to a file we would like to use.
- Have more than one instance of a logger object for different purposes at a given time.
- Use time stamps for logging purposes.
- We can set error levels or masks so the user can reconfigure the system.
- Log the user information for administrative purposes.

Basically there are two methods in the Logger class, *Logit()* to log the events, and *clearLog* for clearing the log. The *MiscDate* class will provide utilities for working with dates and calendars

3.   Html Wrapper

By providing the HtmlWrapper class, the HTML generation process is shifted from the servlet itself to an object whose main function is HTML. In order to be able to do that, the servlet should pass a reference to its output stream to the *HtmlWrapper* object. For Html page generation, template files discussed in the following section, will be used.

a.   *Template File Process*

Template files are used for user input and query output. A user defined language will be used to produce dynamic output without requiring any programming. The language will consist of the following elements: Four tags to differentiate the language from HTML, one attribute, "delim", shared by the tags, a pre-defined vocabulary of identifiers.

When the program processes the template file, all text outside the four predefined tags is left untouched. If a predefined identifier is found between delimiters, then it is replaced by the value of that identifier in the current environment. The entire substitution process assumes that there is always a current environment; the four tags are defined below;

< myThesis:SUBST>: assumes that there is a current environment for substitutions.

< myThesis:SUBSTROW>: takes the next record from the database and makes it current.

<myThesis:SUBSTROWLIST>: runs a query and uses each row of the result as current environment.

<myThesis:SUBSTERR>: is used for outputing error messages.

75

b. *Languages, Grammars and Parsers*

A formal language is a set of strings. A language is made of grammar rules that tell programmers which strings are in the language (grammatical) and which are not (ungrammatical). In the case of a programming language, ungrammatical strings are programs that do not compile.

A formal language defines the following

- vocabulary strings that construct the language (terminal vocabulary)
- another vocabulary to formulate grammar rules (non-terminal vocabulary)
- the rules (also known as productions)
- a start symbol

In the application program, the predefined vocabulary of identifiers will consist of following items

- dbServlet
- dbUser, dbOperation, dbQuery String
- FieldNamel, FieldName2, …
- FieldValuel, FieldValuel 2, …

c. *The Parser*

In a narrow sense, a parser is a program that takes some text as input and produces a data object that represents the syntactic structure of that text, usually a tree. The main components of a parser are

- An input stream or buffer
- A lexical analyzer that obtains the next input token and feeds it into the parser properly
- a parser that attaches the new token to the emerging tree

76

In the application program, the parser will read the template file into a string buffer and construct the *ParseSubst* Object using the buffer and current environment object. The *ParseSubst* object has a *toString()* method that takes an environment as an argument and writes the *ParseSubst* object to string.

4.   Env (Environment) Class

Env class is basically used for data interchange. The first task of the *Env* class is to get the user parameters from the *HttpServletRequest* object using the *getParameterNames* and *getParameterValues* methods. *Env* class has a constructor, requiring *an HttpServletRequest* object as input. The constructor reads the parameter name-value pairs from the request object and constructs the *Env* object.

5.   DBHandler Class

The *DBHandler* class handles interactions with the database.   There is one instance of *DBHandler* for each database, used by the application within a session.

The *DBHandler* consist of two parts; one performs managerial tasks the other contains a hash table of Query objects.   There will be a query object which is defined in the initialization file.   DBHandler will contain the Query as an inner class.

DBHandler class will consist of

- imports, variable declarations
- constructors and an initialize method for the common parts of the constructors
- *getQueryResult()* which returns a result set as a matrix and *getQueryRows()* which returns the result set row by row.
- *Query* inner class
- *addQuery()* method to add a query to hash table and *delQuery()* to delete existing queries
- three methods for connection pooling.

The DBHandler class can handle both SELECT and UPDATE queries. As discussed in Chapter 4, the result set of these queries differ. *executeQuery* returns the selected fields and *executeUpdate* returns an integer number, which is the number of affected rows.

The *Query* objects will be kept in a hash table named *Queries*, indexed by by query names. The current operation parameter will contain the name of the query submitted by the user. This query name will be retrieved from the user's request.

a. Query Class

The Query class implements the "named query" abstraction. It has two constructors that receive query names and text in one or two arrays. The first job of the constructors is to determine whether the query is a SELECT query which returns a result set or an UPDATE query which returns an integer. Next, it counts the number of arguments for the query, by counting the question marks in the query string. Question marks in a Java query string represents parameters to the query. Finally it submits the query string to the *prepareStatement()* method of the connection object.

b. Query Processing

Both the *GetQueryResult()* and *getQueryRows()* methods of the *Query* inner class take *Env* as argument and retrieve a *dbOperation* parameter from it. The parameter *dbOperation* gives the name of the query to be executed. This name is used to retrieve an appropriate *Query* object from the hash table and the corresponding query statement to be run.

The difference between these methods is: *getQueryResult()* returns the result converted to a string matrix. *getQueryRows()* on the other hand returns the actual result set object wrapped inside a *RowSequence*.

Adding and removing of Query objects in the hash table is handled via *addQuery()* and *delQuery()* respectively.

78

6.   Initialization Code

The initialization code of constructors puts all the information about the application (such as database driver, url, user information, query information) into local variables.

As mentioned before, query information(SQL statement) is placed into *Query* objects, which are stored in a hash table kept by the names of the queries. As the initialization code is executed, query SQL statements are checked for white space and trimmed then converted to Query objects to store in the hash table for further use.

a.   *Initialization File*

The configuration parameters are stored in a text file. When the application starts these parameters will contain both keys and values. This initialization file will also contain the SQL queries which are used in the application, and their corresponding names. These names are referred to, whenever required to run a query. A sample initilazitaion file resebles the following:

FileTitle

Sample Thesis Servlet Stok No Search Initialization-Environment File

dbDriver

sun.jdbc.odbc.JdbcOdbcDriver

dbName

jdbc:odbc:THESISSQL

dbQueries

PARTSEARCH,MANSEARCH,REFSEARCH,CHARSEARCH,ADD

PARTSEARCH

SELECT GS,NIIN,NAME,CAGE,INC FROM REFINFO WHERE PARTNO=?

MANSEARCH

SELECT UI,QUP,PRICE,SLC,MGMTCTL FROM MANINFO WHERE NIIN=?

79

REFSEARCH

SELECT GS,NIIN,NAME,PARTNO,CAGE FROM REFINFO WHERE NIIN=?

CHARSEARCH

SELECT GS,NIIN,NAME,CODE,CODEXP FROM CHARACINFO WHERE NIIN=?

ADD

INSERT INTO REFERENCE VALUES(?,?,?)

7. String Splitter Class

By convention query names are separated from each other by a comma. The list of query names is stored in a string variable *dbQuery* in *Env* class. The variable *dbQuery* will be converted to an array of strings, containing query names using a string split utility method.

The *StringSplitter* object provides three methods that allows the delimeter to be a string not only a character comma. If it finds two delimiters in a row, it skips both of them. Unlike *java.util.StringTokenizer*, *StringSplitter* class allows the delimeter to be a string as well as a character.

8. Row Sequence Class

RowSequence class can be considered as an adapter class. It represents a result set returned by a database query in terms of *Environment* objects, later, these objects are sent to *HTMLWrapper* to produce HTML output.

As an adapter class, *RowSequence* class has the ability to extract information from a result set and provide its information to the receiving side in a generic interface. Information extraction is accomplished in the constructors with the help of *MiscDB* class.

9.   MiscDB Class

MiscDB class provides relational database utilities such as column names in the result set, the data types of the fields, and information about the entire Relational Database management System (RDBMS).

THIS PAGE INTENTIONALLY LEFT BLANK

VII. CONCLUSIONS

A.   SYNOPSIS

This thesis presented the design, development and implementation of the Supply Centers Material Request (SMART) database and application. The SMART system will provide Navy personal with an automated system for submitting material requests, and will provide the Supply Centers with an efficient and reliable way of retreiving user requests.

In addition to implementing a prototype database and application program, this thesis also examined the Supply Center's current systems and the need for such an application program. Existing system architectures that best meet the Navy and Supply Centers' needs were profiled in detail. Recent technological advances (as they apply to the selected system architechture) were discussed and samples for dynamic web content developed.

Based on the study findings, middle tier Java Servlet technology was selected to create an application program. Java servlets are used for implementation, and the Java Database Connectivity (JDBC) application program interface (API) was used as a tool for developing an application program.

A conceptual schema was created for the Supply Center's database by using the Semantic Object Modeling technique.

Semantic objects were transformed into a relational model and related tables. The database was created using Microsoft SQL Server 7.0.

User interfaces, including Login page and all the menus, were created using Html and Microsoft Front Page 2000.

Upon completion of the program, the database are populated by entries, and all the menus checked using the existing data.

The main goal of this thesis was to design and implement a component-based three-tiered, web-based system prototype that enables enterprise level applications to scale and exist on different platforms and operating systems. The use of Java and Java servlets which come free and provide component based development architecture, enabling the use of previously created componets, lowers the cost of development and maintenance.

The prototype demonstrated how to overcome the stateless nature of HTTP using the session tracking API of Java servlets. Also, the prototype provided a faster way to access enterprise databases using the connection pools.

The source code for the application prototype can be tailored to specific business requirements, and may also be used to build up three tier applications for any use. It is hoped that this system, as an initial effort, will promote further efforts to develop new systems that will benefit other branches of the Turkish Navy, and inspire the creation of other dedicated, focused systems.

B.   FUTURE ENHANCEMENTS

1.   Java Server Pages (JSP)

In the application program designed to create HTML output, template files were used in conjuction with a user-defined grammer. While this thesis research was in progress, a technological innovation, Java Server Pages (JSP), were introduced. JSP enables programmers to design and format the result page using HTML or extensible markup language (XML), and dynamic content can be generated by using JSP tags or scriplets. The program logic can be encapsulated in tags and JavaBeans components, and tied together in scriptlets that work on the server side.

If the program logic can be encapsulated in JSP tags and Java Beans, HTML pages can be designed by webmasters without effecting the generation of the content [Ref.20].

2.   Extensible Markup Language (XML)

XML is a markup language adopted by the World Wide Web Consortium (W3C) in December 1999. XML differs from HTML in that HTML has a predefined number of tags, and anything tags other than these are simply ignored by the browser, whereas XML has only a few predefined tags, and gives the programmer power to develop his own tags. By using XML, a parser capable of reading an XML file can access data and meta-data represented by the programmer using his own tags. In the case of HTML, the parser, which is actually the web browser, will translate these tags to a representation without addressing the meta-data.

With the introduction of XML to the application program, it would be possible to use Extensible Style Sheet (XSL) to view and represent XML content. With the use of XSL with XML, for any single XML document, there could be many XSL schemas to display the document on a browser. For example, one XSL can be used to display data sorted by name, another one can sort the data by a last name, amd a third one could be used to display data on a Personal Data Assistant and cellular phone.

All the text files used for setup purposes in the application program could be transformed into XML files. The use of XML files will enable the program to check the setup files for proper syntax and validation against Document Type Definition (DTD), which represents the structure of XML file. DTD files can be found in any location which is accessible by network, giving the program extra scalability.

In summary, the introduction of XML would provide the application program with:

Flexibilty: XML supports user defined tags while enforcing structure and correctness of the XML file.

Reusability: XML will separate configuration data from the working code, making it possible to obtain a reusable program that interact with many XML files.

85

Standard: XML is a W3C standard, which is non-proprietary and universally accepted.

Interoperability and Portability: An XML document will behave the same way on any platform running any known COTS operating system. This characteristic enables programmers to store XML files on any platform and access those files using a Unified Resource Identifier (URI).

3. Additional Security

As discussed in Chapter V, all the users accessing the application program are assumed to have the same user ID and password couple. In order to increase security, each user can be assigned a security level, and individual connection pools can be created for each security level.

APPENDICES

APPENDIX A. DATA.HTML FILE

```html
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<!-- CGI Sample : data.html -->
<HTML>
<HEAD>
<TITLE>Sample Database Query</TITLE>
</HEAD>
<BODY BACKGROUND = "images/back.gif">
<img border="0" src="images/leaf.gif" width="75" height="69">
<p>
<BASEFONT FACE = "ARIAL,SANS-SERIF" SIZE = 2>
  <FONT SIZE = +2>
    <STRONG>Querying Supply Center's database, Using CGI-Perl and
ODBC.</STRONG>
  </FONT><BR>
</p>
      <p>
      <BASEFONT FACE = "ARIAL,SANS-SERIF" SIZE = 2>
    <font size="3" color="#FF0000">
    <b>Enter The Query String</b>
    </font>
    </p>
  <FORM METHOD = "POST" ACTION = "cgi-bin/data.pl">
  <p>
  <INPUT NAME = "QUERY" SIZE = 40
      VALUE = "SELECT * FROM REQUEST">
  </p>
  <p>
  <font size="3" color="#FF0000">
  <b>Press The Button To Submit The Query</b>
  </font>
  </p>
  <p>
  <INPUT TYPE = "SUBMIT" VALUE = "Send Query">
  </p>
  </FORM>
  <p>
  <img     border="0"     src="../thesis/images/mailyellow.gif"     width="70"
height="50"></p>
  </BODY>
  </HTML>
```

APPENDIX B. DATA.PL FILE

```perl
# CGI Perl Sample: data.pl
# Program to query a database and send
# results to the client.
use Win32::ODBC;
use CGI qw/:standard/;
my $querystring = param(QUERY);
$DSN = "THESISSQL";
print header;
if (!($Data = new Win32::ODBC($DSN)))
{
  print "Error connecting to ".$DSN." "\n";
  print "Error: " . Win32::ODBC::Error() . "\n";
  exit;
}
if ($Data->Sql($querystring))
{
  print "SQL failed.\n";
  print "Error: " . $Data->Error() . "\n";
  $Data->Close();
  exit;
}
print "<BODY BACKGROUND = \"/images/back.gif\">";
print "<BASEFONT FACE = \"ARIAL,SANS-SERIF\" SIZE = 3>";
print "<FONT COLOR = BLUE SIZE = 4> Search Results </FONT>";
$counter = 0;
print "<TABLE BORDER = 0 CELLPADDING = 5 CELLSPACING = 0>";
while($Data->FetchRow())
{
  %Data = $Data->DataHash();
  @key_entries = keys(%Data);
  print "<TR>";
  foreach $key( keys( %Data ) )
  {
    print "<TD BGCOLOR = #9999CC>$Data{$key}</TD>";
  }
  print "</TR>";
  $counter++;
}
print "</TABLE>";
print "<BR>Your search yielded <B>$counter</B> results.";
print "<BR><BR>";
print "<FONT SIZE = 2>";
print "Please email comments to ";
```

```
print "<A href = \"mailto:Administor\@cem-home.com\">Cemalettin </A>.";
print end_html;
$Data->Close();
```

APPENDIX C. LOGIN.ASP FILE

```
<% @LANGUAGE=VBScript %>
<% Option Explicit %>
<% ' ASP sample: login.asp %>
<%
    ' Set up the variables for this page
    Dim dbConn, dbQuery, loginRS, loginFound
    ' Check to see if there is an existing connection to
    ' the Database. If not, create one
    If IsObject( Session( "thesissql_dbConn" ) ) Then
        Set dbConn = Session( "thesissql_dbConn" )
    Else
        Set dbConn = Server.CreateObject( "ADODB.Connection" )
        Call dbConn.Open( "thesissql", "", "" )
        Set Session( "thesissql_dbConn" ) = dbConn
    End If
    ' Create the SQL query
    dbQuery = "SELECT * FROM users"
    ' Create the recordset
    Set loginRS = Server.CreateObject( "ADODB.Recordset" )
    Call loginRS.Open( dbQuery, dbConn )
    On Error Resume Next ' If an error occurs, ignore it
    ' Move to the first record in the recordset
    Call loginRS.MoveFirst()
%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD><TITLE>Login Page</TITLE></HEAD>
<BODY>
<!--        include        header        goes        here--><!--        #include
virtual="aspSample/includes/thesisHeader.inc" --><%
    ' If this is a return after a failed attempt, print an error
    If Session( "loginFailure" ) = True Then %>
<FONT SIZE = 4 COLOR = "red"> Login attempt failed,
please try again <P></FONT>
    <% End If %>
<% ' Begin the form %>
<font color="#FF0000"> </font>
<FONT FACE = "arial" SIZE = 2>
<font color="#FF0000">Please select your name and enter
your password to login:</font><BR>
</FONT>
```

```
        <FORM NAME = sublogform ACTION = "submitlogin.asp" METHOD =
POST>
        <% ' Format the form using a table %>
        <TABLE BORDER = 0>
        <TR>
        <TD><FONT     FACE     =     "arial"     SIZE     =     2
color="#0000FF">Name:</FONT></TD>
        <TD><SELECT NAME = "LOGINID">
        <OPTION VALUE = "000">Select your name
        <%
          ' Pull user names from the query to populate the dropdown
          While Not loginRS.EOF
            ' If there is a session loginid, reuse it
            If Session( "loginid" ) = loginRS( "loginid" ) Then
              loginFound = "selected "
            End If
            ' If a login cookie was found, reuse it
            If Request.Cookies( "loginid" ) = loginRS( "loginid" ) Then
              loginfound = "selected "
            End If
            ' Create each dropdown entry %>
        <OPTION <% =loginFound %>
        value="<% =loginRS( "loginid" ) %>">
        <% =loginRS( "loginid" ) %>
        <% loginfound = " " %>
        <%
            Call loginRS.MoveNext()
          Wend
        %>
        </SELECT></TD></TR>
        <TR><TD><FONT     FACE     =     "arial"     SIZE     =     "2"><font
color="#0000FF">Password</font>:</FONT></TD>
        <TD><INPUT TYPE = "password" NAME = "SUBMIT_LOGIN"></TD></TR>
        <TR><TD> </TD>
        <TD ALIGN = "LEFT"><INPUT TYPE = "submit" VALUE = "Log Me In"
ID = "login1" NAME = "login1"></TD></TR>
        </TABLE></FORM>
        <!-- #include virtual="aspSample/includes/thesisFooter.inc" -->
        </BODY>
        </HTML>
```

APPENDIX D. SUBMITLOGIN.ASP FILE

```
<% @LANGUAGE=VBScript %>
<% Option Explicit %>
<% ' aspSample: submitlogin.asp %>
<%
  ' Set up the variables for this page
  Dim dbConn, dbQuery, loginRS
  ' Check to see if there is an existing connection to
  ' the Database.  If not, create one
  If IsObject( Session( "thesissql_dbConn" ) ) Then
    Set dbConn = Session( "thesissql_dbConn" )
  Else
    Set dbConn = Server.CreateObject( "ADODB.Connection" )
    Call dbConn.Open( "thesissql", "", "" )
    Set Session( "thesissql_dbConn" ) = dbConn
  End If
  ' Create the SQL query
  dbQuery = "SELECT * FROM users"
  ' Create the recordset
  Set loginRS = Server.CreateObject( "ADODB.Recordset" )
  Call loginRS.Open( dbQuery, dbConn )
  On Error Resume Next   ' If an error occurs, ignore it
  ' Move to the first record in the recordset
  Call loginRS.MoveFirst()
  ' If the loginid is not empty then
  If Request( "loginid" ) <> "" Then
    While Not loginRS.EOF
      If Request( "loginid" ) = loginRS( "loginid" ) AND _
        Request( "submit_login" ) = loginRS( "password" ) Then
        ' Password and loginid are OK set a Session variable
        Session( "loginfailure" ) = False
        ' Set a cookie to recognize them the next time they
        ' go to login.asp
        Response.Cookies( "loginid" ) = Request( "loginid" )
        ' Send them on to the next page
        Call Response.Redirect( "instantpage.asp" )
      End If
        Call loginRS.MoveNext()    ' Move on to the next record
    Wend
  End If
  ' If loginid is empty, or no match was found
  Session( "loginFailure" ) = True  ' Set loginFailure to true
  ' Return to the login page
  Call Response.Redirect( "login.asp" )%>
```

APPENDIX E. THESISSERVLET.JAVA

```java
import javax.servlet.*;  // communicate with client
import javax.servlet.http.*;
import mythesis.utilityClasses.Logger;  // saves admin/debug info to file
import mythesis.utilityClasses.Env;   // basic package
import mythesis.utilityClasses.MiscFile;   // basic package
import mythesis.utilityClasses.DBHandler;   // communicate with database
import mythesis.utilityClasses.RowSequence;   // database results
import java.sql.SQLException;
import mythesis.utilityClasses.HtmlWrapper; // sends HTML to client.
import java.io.IOException; // thrown by HtmlWrapper
import mythesis.utilityClasses.MiscDate; // for logging.
// it starts "InitDBHandler"
// a new session containing a DBHandler initialized with the Environment
provided by
// the servlet request; it is aware That LOGOUT means it should kill the session.
All the
// Information about what queries are to be used, and how to present the results,
//comes from initialization file: in this case, the queries are defined in the
//initDBHandler html
// form, and each query can refer to an html template file for output.

public class ThesisServlet extends HttpServlet {
final String filePath = "myThesis/ThesisServlet/";
// D:\Jsdk2.0\MyThesis\ThesisServlet
final String iniFileName = "ThesisServlet.ini";    // defaults for servletRequest
final String topFileName = "thesistop.htm";  // or override with getInitParam
final String ctlFileName = "thesisctl.htm";
Logger lg;
public void doGet (HttpServletRequest req,
            HttpServletResponse res)
      throws ServletException, IOException{
  doPost(req,res);
}
public void doPost (HttpServletRequest req,
            HttpServletResponse res)
      throws ServletException, IOException{
  res.setContentType("text/html");
  // Create an HtmlWrapper object
  HtmlWrapper W=new HtmlWrapper(res.getWriter());
  // Create a session object
  HttpSession sess=req.getSession(true);
  lg=new Logger();
  try{
```

93

```java
    // Create an Env object from the request object.
    Env E=new Env(req);
    //add the URL of the servlet to Env
    String myURL=res.encodeUrl(req.getRequestURI());
    E.put("dbServlet",myURL);
    String dbOperation=E.getStr("dbOperation");
  // If the value of dbOperation is InitDBHandler, read .ini file, get username and
// password from request object and create a DBHandler
    if("InitDBHandler".equals(dbOperation))lg.clearLog();
    if("InitDBHandler".equals(dbOperation))doInit(sess,E,W);
    else if(dbOperation==null)sendCtl(sess,E,W);
    // if dbOperation is Logout invalidate the session.
    else if("Logout".equals(dbOperation))doEnd(sess,W);
    // if the dbOperation is a query name , run the specified query and
    // return the result.
    else doQuery(sess,E,W);
   }catch(Exception ex){
    W.wrapPage("doPost failure",Logger.stackTrace(ex));}
  }
 public void doInit(HttpSession sess,Env E,HtmlWrapper W)
   throws SQLException,Exception {
   // set the value of file path
  String fP=setStr(E,"filePath","",filePath);
  if(!fP.endsWith("/"))E.put("filePath",fP+="/");
  sess.putValue("filePath",fP);
   // find the initialization file
  String ini=setStr(E,"iniFileName",fP,iniFileName);
  // add initialization file content to Env object
  E.addBufferedReader(MiscFile.getBufferedReader(ini));
  // add template file name and value pair to Env.
  setStr(E,"templateFile",fP,"topFileName",topFileName);
  String myURL=E.getStr("dbServlet");
  if(myURL.indexOf('?')<0)
      //URL must have query arguments
   E.put("dbServlet",myURL+"?dummyField=dummyVal");
  sess.putValue("theDBHandler",new DBHandler(E));
  W.wrapEnvPage(E); // the Env defines output
 }
// send the control page dbOperation==null
public void sendCtl(HttpSession sess,Env E, HtmlWrapper W){
// normally used to fill in a frame, but as a response
// to null query this is a usage message.
 String fP=(String)sess.getValue("filePath");
 setStr(E,"templateFile",fP,"ctlFileName",ctlFileName);
```

94

```java
    lg.logIt("sendCtl(filePath="+fP+"): "+E.toString());
     W.wrapEnvPage(E);
    }
   //execute the queries requested by the user
  public void doQuery(HttpSession sess,Env env,HtmlWrapper W)
       throws SQLException,IOException {
  // process user query
  lg.logIt("doQuery: "+env.toString());
    String fP=(String)sess.getValue("filePath");
    DBHandler dbH=(DBHandler)sess.getValue("theDBHandler");
    if(dbH==null)
      W.wrapPage("doPost Failure","No dbhandler in sess "+sess.getId());
    else {
      setStr(env,"templateFile",fP,env.getStr("dbOperation")+".htm");
      RowSequence rows=dbH.getQueryRows(env);
      W.wrapRowsPage(rows,env); // again the Env defines output.
      }
    dbH.gotoSleep();
   }
  // end the session
  public void doEnd(HttpSession sess, HtmlWrapper W)
     throws IOException,SQLException {
  lg.logIt("doEnd sess "+sess.getId());
    DBHandler dbH=(DBHandler)sess.getValue("theDBHandler");
    if(dbH!=null)dbH.close();
    sess.invalidate();
    W.wrapPage("Session Ends","come back soon");
    }
  public String setStr(Env E,String resNm,String pre,String nm,String dflt){
    //sets E[resNm] to pre+(getInitParameter(nm) or E[nm] or dflt)
    String val=getInitParameter(nm);
    if(null==val)val=E.getStr(nm);
    if(null==val)val=dflt;
    E.put(resNm,pre+val);
    return pre+val;
  }
  public String setStr(Env E,String nm,String pre,String dflt){
    //sets E[nm] to pre+(getInitParameter(nm) or E[nm] or dflt)
    return setStr(E,nm,pre,nm,dflt);
  }}
```

95

APPENDIX F. LOGGER.JAVA FILE

```java
package mythesis.utilityClasses;
import java.io.*;
public class Logger { // may be shared across threads.
  static String fileProp="mythesis.utilityClasses.Logger.file";
  static String debugProp="mythesis.utilityClasses.Logger.debugLevel";
  static String defaultFileName="dbLog.log";
  String fileName;
  int debugLevel;
public Logger(String fName){
  fileName=fName;
  debugLevel=Integer.parseInt(System.getProperty(debugProp,"1"));
  }
public Logger(){this(System.getProperty(fileProp,defaultFileName));}
public synchronized void setFileName(String fName){fileName=fName;}
public synchronized void setDebugLevel(int N){debugLevel=N;}
public synchronized void setDebug(boolean B){debugLevel=B?1:0;}
public synchronized void setDebug(String S){
  setDebug((new Boolean(S)).booleanValue());
}
public synchronized void clearLog(){
  if(debugLevel<=0)return;
  try{
   PrintWriter f=new PrintWriter(new FileWriter(fileName,true));
   String S=MiscDate.todaysDate();
   f.println(S);
   f.close();
  }catch(IOException e){}
}
public synchronized void logIt(String S){
  if(debugLevel<=0)return;
  try{
   PrintWriter f=new PrintWriter(new FileWriter(fileName,true));
   f.println(S);
   f.close();
  }catch(IOException e){}
}
public synchronized void logIt(String S,Throwable ex){
  if(debugLevel<=0)return;
  try{
   PrintWriter f=new PrintWriter(new FileWriter(fileName,true));
   f.println(S);
   ex.printStackTrace(f);
   f.close();
```

```java
        }catch(IOException e){}
   }
   public static String stackTrace(Throwable ex){
      StringWriter sw=new StringWriter();
      ex.printStackTrace(new PrintWriter(sw));
      return sw.toString();
   }
}
```

APPENDIX G. ENV.JAVA

```java
package mythesis.utilityClasses;
import java.util.Hashtable;
import java.util.Stack;
import java.util.Enumeration;
import java.io.Serializable;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.FileNotFoundException;
import java.io.FileReader;
import javax.servlet.http.HttpServletRequest;

public class Env extends Hashtable {
Logger lg;
  // keys and values to be strings or string-arrays;
public Env(Hashtable H){
  lg=new Logger();
  addHashtable(H);
}
public void addHashtable(Hashtable H){
  Enumeration k=H.keys();
  while(k.hasMoreElements()){
    Object S=k.nextElement();
    this.put(S,H.get(S));
    }
  addEnvironmentFile();
}
public String toString(){
  StringBuffer sB=new StringBuffer();
  Enumeration k=keys();
  while(k.hasMoreElements()){
    String S=(String)k.nextElement();
    sB.append(S);
    sB.append("=");
    sB.append(getStr(S));
    sB.append(", \t");
    }
  return sB.toString();
}
public String toStringRec(){
  // use if subEnvs might be cyclic
  StringBuffer sB=new StringBuffer();
  tSS(this,new Stack(),sB);
  return sB.toString();
```

```
  }
public void tSS(Env E,Stack eS,StringBuffer sB){
  // tostring subfunction for safe recursive Envs
  if(0<=eS.search(E)){
    sB.append("***CYCLIC ENV***\n");
    return;
    }
  eS.push(E);
  Enumeration k=E.keys();
  while(k.hasMoreElements()){
    String key=(String)k.nextElement();
    for(int i=0;i<eS.size();i++)sB.append("  ");
    sB.append(key); sB.append("=");
    Object ob=E.get(key);
    if(null==ob)sB.append("null;\n");
    else if(ob instanceof String){
      sB.append((String)ob);
      sB.append(";\n");
      }
    else if(ob instanceof String[]){
      sB.append(Misc.stringArrayJoin((String[])ob,", "));
      sB.append(";\n");
      }
    else if(ob instanceof Env){
      sB.append("[\n");
      tSS((Env)ob,eS,sB);
      for(int i=0;i<=eS.size();i++)sB.append("  ");
      sB.append("]\n");
      }
    else sB.append("??\n");
  }
  eS.pop();
}
public Env(){
        lg=new Logger();
}
public Env(BufferedReader brin){
  lg=new Logger();
  try{
  addBufferedReader(brin);
  }catch(Exception ex){lg.logIt("env br fail: "+ex);}
}
public String getLine(BufferedReader brin){
  try{
```

```java
        String theLine=brin.readLine();
        if(theLine==null)return null;
        else return theLine;
      }catch(Exception ex){return null;}
    }

    public    void    addBufferedReader    (BufferedReader    brin)throws
ParseSubstException {
      if(brin==null){
        lg.logIt("addBufferedReader failure: null reader");
        return;
        }
      String key; String line; StringBuffer value=new StringBuffer();
      try{
      while((key=brin.readLine())!=null){
       if((line=brin.readLine())!=null){
         value.setLength(0);
         while(line!=null && line.endsWith("\\") && !line.endsWith("\\\\")){
           value.append(line);
           value.setCharAt(value.length()-1,'\n');
           line=brin.readLine();
           }
         if(line!=null)value.append(line);
         putQuotedVal(key,Misc.substLineByTags(value.toString(),this));
         }
        }
      brin.close();
      addEnvironmentFile();
      }
      catch(IOException ex)
        {lg.logIt("addBufferedReader: ",ex);}
    }
    public void addEnvironmentFile(){
      String envFile=getStr("includeEnvironmentFile");
      if(envFile==null)return;
      lg.logIt("including env file "+envFile);
      try{
      remove("includeEnvironmentFile");
      put("includedEnvironmentFile",envFile);
      BufferedReader br=getBufferedReader(envFile);
      if(br==null){lg.logIt("null envFile br"); return;}
      addBufferedReader(br);
      lg.logIt("included env file "+envFile);
      }catch(Exception ex){lg.logIt("addEnvF:"+ex);}
```

100

```java
  }
  public void putQuotedVal(String key,String val){
    val=Misc.evalQuotedChars(val);
    if(!key.startsWith("[]"))put(key,val);
    else put(key.substring(2,key.length()),
          Misc.stringSplit(val) );
  }
  public Env(String fName){
    this(getBufferedReader(fName));
  }
  public static BufferedReader getBufferedReader(String fName){
   try{
    FileReader fR=new FileReader(fName);
    if(fR==null)return null;
    return new BufferedReader(fR);
   }catch(Exception ex)
     {ex.printStackTrace();
      return null;}
  }
  public Env(HttpServletRequest req){
   lg=new Logger(); lg.logIt("reading an httpservletrequest");
   Enumeration E=req.getParameterNames();
   while(E.hasMoreElements()){
    String name=(String)E.nextElement();
    String[]vals=req.getParameterValues(name);
    if(vals.length!=1)this.put(name,vals);
    else this.put(name,vals[0]);
    }
   lg.logIt("got through initial env");
   addEnvironmentFile();
  }
  public String getStr(String key){
   Object ob=this.get(key);
   if(ob==null)lg.logIt("null getStr for ["+key+"]");
   if(ob instanceof String)
    return (String)ob;
   else if(ob instanceof String[])
    return Misc.stringArrayJoin((String[])ob,", ");
   else if(ob instanceof Env)
    return "["+((Env)ob).toString()+"]";
   else return null;
  }
  public String getStr(String key,String dflt){
   Object ob=this.get(key);
```

101

```java
    if(ob==null)return dflt;
    if(ob instanceof String) return (String)ob;
    else if(ob instanceof String[])
      return Misc.stringArrayJoin((String[])ob,", ");
    else return dflt;
  }
  public String[] getStrSeq(String key){
    Object ob=this.get(key);
    if(ob==null)lg.logIt("null getStrSeq for ["+key+"]");
    if(ob instanceof String[]) return (String[])ob;
    else if(ob instanceof String) return new String[]{(String)ob};
    else return null;
  }
  public int getInt(String key,int dfault){
    Object ob=this.get(key);
    if(ob==null)return dfault;
    if(ob instanceof String) return Integer.parseInt((String)ob);
    return ((Integer)ob).intValue();
  }
  public void append2StrSeq(String key,String val){
    Object ob=get(key);
    if(null==ob)return;
    String[]A;
    if(ob instanceof String[])A=(String[])ob;
    else if(ob instanceof String)A=new String[]{(String)ob};
    else return;
    String[]B=new String[A.length+1];
    for(int i=0;i<A.length;i++)B[i]=A[i];
    B[A.length]=val;
    put(key,B);
  }
  public void takeStrSeq(String key,int numToTake){
    Object ob=get(key);
    if(null==ob)return;
    String[]A;
    if(ob instanceof String[])A=(String[])ob;
    else if(ob instanceof String)A=new String[]{(String)ob};
    else return;
    if(numToTake>=A.length)return;
    String[]B=new String[numToTake];
    for(int i=0;i<numToTake;i++)B[i]=A[i];
    put(key,B);
  }
  public Env getEnv(String key){
```

```java
    Object ob=get(key);
    if(ob==null || !(ob instanceof Env))return null;
    return (Env)ob;}
  public void put(String[]keys,Object val){
   if(null==keys||0==keys.length)return;
   Env E=this; int lastIndex=keys.length-1;
   for(int i=0;i<lastIndex;i++){
     Object ob=E.get(keys[i]);
     if(null==ob || !(ob instanceof Env))
       {ob=new Env(); E.put(keys[i],ob);}
     E=(Env)ob;    }
   E.put(keys[lastIndex],val);}
  public Object get(String[]keys){
   if(null==keys||0==keys.length)return null;
   Env E=this; int lastIndex=keys.length-1;
   for(int i=0;i<lastIndex;i++){
     Object ob=E.get(keys[i]);
     if(null==ob || !(ob instanceof Env)) return null;
     E=(Env)ob;    }
   return E.get(keys[lastIndex]);}
  public void putSplit(String key, Object val){
   if(null==key)return;
   if(key.indexOf('_')<0)put(key,val);
   else put(Misc.stringSplit(key,'_'),val);}
  public Object getSplit(String key){
   if(null==key)return null;
   if(key.indexOf('_')<0)return get(key);
   return get(Misc.stringSplit(key,'_'));
  }
}
```

APPENDIX H. DBHANDLER.JAVA

```java
package mythesis.utilityClasses;
import java.util.Hashtable;
import java.sql.Date;
import java.text.DateFormat;
import java.util.Enumeration;
import java.sql.*;   // for communicating with database
public class DBHandler  {
  DBConnectionManager dbCM=null; //one Cache of connection pools
  DBConnectionPool dbCP=null;    // connection pool for this dbUrl/usr/pwd
  Connection theConnection=null;
  static String defaultDateFormat="yyyy-MM-dd"; // for month,day,year
  String dateFormat=null;
  java.text.SimpleDateFormat simpleDateFormat; // reads datestrings
  Hashtable theQueries=null; // contains prepackaged queries
  String currentOp=null;
  String driverName=null; // ="sun.jdbc.odbc.JdbcOdbcDriver";
  String dbUrl=null;  // ="jdbc:odbc:THESISSQL";
  String theUser=null; // ="usr";
  String thePwd=null; // ="pwd";
  Logger lg;
  // constructor 1 receives all the string values as individual arguments
  // and calls the initDBHandler() method
  public DBHandler(String dbDriver,String dbName,
            String dbUser,String dbPwd,
            String [] qNames,String [] qVals)
              throws SQLException{
    lg=new Logger();
    initDBHandler(dbDriver,dbName,dbUser,dbPwd,qNames,qVals,null);
  }
  // constructor 2 receives all the string values as individual arguments
  // and calls the initDBHandler() method

  public DBHandler(String dbDriver,String dbName,
            String dbUser,String dbPwd,
            String [] qNames,String [] qVals,
            String [] qTypes)
              throws SQLException{
    lg=new Logger();
    initDBHandler(dbDriver,dbName,dbUser,dbPwd,qNames,qVals,qTypes);
  }
  // constructor 3 receives all the string values from ENV object
  // and calls the initDBHandler() method
  public DBHandler(Env env)throws SQLException,Exception{
```

```java
      lg=new Logger();
      // get the values from env object
      String dbDriver=env.getStr("dbDriver");
      String dbName=env.getStr("dbName");
      String dbUser=env.getStr("dbUser");
      String dbPwd=env.getStr("dbPwd");
      String dbQueries=env.getStr("dbQueries"); // Queries that we are going to use
      dateFormat=env.getStr("dateFormat");
      // dbQueries is a comma seperated list of query names
      // convert it to an array of strings by utility method Misc.stringSplit()
      String [] qNames=Misc.stringSplit(dbQueries,',');
      env.put("dbQueries",qNames); // it is a StrSeq.
      String [] qVals=new String[qNames.length];
      String [] qTypes=new String[qNames.length];
      for(int i=0;i<qVals.length;i++){
        qVals[i]=Misc.substLineByTags(env.getStr(qNames[i]),env);
        qTypes[i]=env.getStr(qNames[i]+"_types");
        }
      initDBHandler(dbDriver,dbName,dbUser,dbPwd,qNames,qVals,qTypes);
       }
   // check for the connection
   protected Connection checkConnection()throws SQLException{
     if(null!=theConnection)return theConnection;
     try{

if(null==dbCM)dbCM=(DBConnectionManager)DBConnectionManager.getInstance();
     dbCM.addDriver(driverName);
     if(null==dbCP)dbCP=dbCM.getConnectionPool(dbUrl,theUser,thePwd);
     theConnection=dbCP.getConnection();
     if(null==theConnection)
       throw new SQLException(dbUrl+", driver "+driverName + " null connect");
     lg.logIt("DBHandler got connection for "+dbUrl+", "+theUser);
     return theConnection;
     }catch(Exception ex){
     lg.logIt("DBHandler.checkConnection for "+dbUrl+": "+ex);
     throw new SQLException(dbUrl+", driver "+driverName+
                 " failed to connect "+ex);
     }}
   protected void freeConnection(){ // called on close or gotosleep.
     if(null==theConnection)return;
     dbCP.freeConnection(theConnection);
   lg.logIt("DBHandler freed connection for "+dbUrl+", "+theUser);
     theConnection=null;
   }
```

```java
// puts all the information about the application into
// DBHandler variables, al queries are made into Query object and put into hash
table.
    public void initDBHandler(String dbDriver,String dbName,
            String dbUser,String dbPwd,
            String [] qNames,String [] qVals,String[] qTypes)
            throws SQLException{
  driverName=dbDriver;
  dbUrl=dbName;
  theUser=dbUser;
  thePwd=dbPwd;
  theQueries=new Hashtable();
  if(null==dateFormat)dateFormat=defaultDateFormat;
  try{
  lg.logIt("driverName="+driverName+
      "\ndbUrl="+dbUrl+
      "\ntheUser="+theUser+
      "\nthePwd="+thePwd);
   checkConnection();// connection pooling code
  Class.forName(driverName);
  // Connection metadata may be useful in debugging.
  lg.logIt("got the class forName "+driverName);
  theConnection=DriverManager.getConnection(dbUrl,theUser,thePwd);
  lg.logIt("got the connection to "+dbUrl);
  DatabaseMetaData dmd=theConnection.getMetaData();
  lg.logIt("max connections="+dmd.getMaxConnections());
  lg.logIt("max statements="+dmd.getMaxStatements());
  if(qTypes==null)qTypes=new String[qVals.length];
  for(int i=0;i<qNames.length;i++){
        // trim the whitespace around the parameters.
   qVals[i]=qVals[i].trim();
   // create Query objects
   Query Q=new Query(qNames[i],qVals[i],qTypes[i]);
   theQueries.put(qNames[i],Q);
   }
  simpleDateFormat=new    java.text.SimpleDateFormat(dateFormat);   //default
locale
 }catch(Exception ex){
  ex.printStackTrace();
  lg.logIt("DBHandler failed ",ex);
  } }
 public DBHandler(String qSpecStr)throws SQLException{
 lg=new Logger();
 theQueries=new Hashtable();
```

```java
      String[]S=Misc.stringSplit(qSpecStr);
      String[][]qSpec=new String[S.length][];
      for(int i=0;i<S.length;i++)qSpec[i]=Misc.stringSplit(S[i]);
      driverName=qSpec[0][0]; dbUrl=qSpec[0][1];
      theUser=qSpec[0][2]; thePwd=qSpec[0][3];
      lg.logIt("driverName="+driverName+
           "\ndbUrl="+dbUrl+
           "\ntheUser="+theUser+
           "\nthePwd="+thePwd);
    try{
      checkConnection();
     for(int i=1;i<qSpec.length;i++){
      Query Q=new Query(qSpec[i]);
      theQueries.put(qSpec[i][0],Q);  }
     }catch(Exception ex){
      ex.printStackTrace();
      lg.logIt("DBHandler failed with",ex);
      return;
      }
     lg.logIt("DBHandler connected to "+dbUrl);
    }
    public Env getQueryResult(Env qInfo)throws SQLException{
     qInfo.put("dbUser",theUser);
     qInfo.put("dbHandler",this);
     currentOp=qInfo.getStr("dbOperation");
     if(null==currentOp)return null;
     Query Q=(Query)theQueries.get(currentOp);
     return Q==null?null:Q.getQueryResult(qInfo);
    }
    public RowSequence getQueryRows(Env qInfo)throws SQLException{
     qInfo.put("dbUser",theUser);
     qInfo.put("dbHandler",this);
     currentOp=qInfo.getStr("dbOperation");
     if(null==currentOp)
       throw new SQLException("no dbOperation");
     Query Q=(Query)theQueries.get(currentOp);
     if(null==Q)
       throw new SQLException("undefined dbOperation: "+currentOp);
     return Q.getQueryRows(qInfo);
    }
    public void gotoSleep()throws SQLException{
     Enumeration qq=theQueries.elements();
     while(qq.hasMoreElements()){
      Query Q=(Query)qq.nextElement();
```

107

```
      if(null!=Q)Q.close();
      }
    freeConnection();
    }
   public void close()throws SQLException{
    gotoSleep();
    dbCM.freeInstance();
    dbCP=null;
    dbCM=null;
    }
   public void addQuery(String qNm,String qStr,String qT)
          throws SQLException{
     theQueries.put(qNm,new Query(qNm,qStr,qT));
    }
   public void delQuery(String qNm)throws SQLException{
     Query Q=(Query)theQueries.get(qNm);
     if(null==Q)return;
     Q.close();
     theQueries.remove(qNm);
    }
 private class StrSeqList {
  public String [] hd; public int loc; public StrSeqList tl;
  public StrSeqList(String [] h, int l, StrSeqList t){
    hd=h; loc=l; tl=t;}
 }
 private class Query {
  public String qName; public StrSeqList theStrSeqList;
  public String qString;
  public String[]qTypes;
  public PreparedStatement pStmnt=null;
  public int argCount; public int colCount;
  public boolean givesResultSet=false;
  ResultSet theResult=null;
  public void close()throws SQLException{
   if(null!=theResult)theResult.close();
   theResult=null;
   if(null!=pStmnt)pStmnt.close();
   pStmnt=null;
  }
 public Query(String [] Q)throws SQLException{this(Q[0],Q[1],Q[3]);}
 public Query(String qNm, String qStr,String qT) throws SQLException{
   qName=qNm; qString=upcaseQueryString(qStr);
   if(null==qT)qT="";
   qTypes=Misc.stringSplit(qT,',');
```

108

```java
      givesResultSet=qString.startsWith("SELECT");
      argCount=0;
      for(int i=0;i<qStr.length();i++)
        if(qStr.charAt(i)=='?')argCount++;
      colCount=0;
   }
  private PreparedStatement checkPstmnt() throws SQLException{
    if(null==pStmnt)pStmnt=checkConnection().prepareStatement(qString);
    return pStmnt;
  }
  public String upcaseQueryString(String qStr){
    // puts SQL operator, e.g. "Select", into uniform upper case.
    if(null==qStr)return null;
    StringBuffer sB=new StringBuffer(qStr);
    char c;
    for(int i=0; i<sB.length() && Character.isLetter(c=sB.charAt(i)); i++)
      sB.setCharAt(i,Character.toUpperCase(c));
    return sB.toString();
  }
  public Env getQueryResult(Env qInfo) throws SQLException{
// using an Env to receive  parameters needed, then to produce results
// The input and result Env are the same object;
// a resultset is returned as "ResultTable", a 2-D matrix
  checkPstmnt();
  argCount=1; String V;
  int maxArgs=qInfo.getInt("ParameterMax",1000);
  while (argCount<=maxArgs &&
      null!=(V=qInfo.getStr("Parameter"+argCount))){
    setParamStr(argCount++,V);
    }
  qInfo.put("NumberOfParameters",""+(argCount-1));
  qInfo.put("dbQueryString",qString);
  if (givesResultSet){
    theResult=pStmnt.executeQuery();
    qInfo.put("ResultTable", MiscDB.resultRowsToStringMatrix(theResult));
    theResult.close();
    return qInfo;
    }
  else {int N=pStmnt.executeUpdate();
    qInfo.put("NumberOfRowsAffected",""+N);
    return qInfo;
   } }
  public void setParamStr(int i,String val)throws SQLException{
   try{
```

```java
      String t=(i>qTypes.length)?null:qTypes[i-1];
      if(t==null||"text".equalsIgnoreCase(t)
       ||"varchar".equalsIgnoreCase(t)||"longvarchar".equalsIgnoreCase(t))
        pStmnt.setString(i,val);
      else if(t.equalsIgnoreCase("date")){
        java.util.Date d=simpleDateFormat.parse(val);
        java.sql.Date dbdate=new java.sql.Date(d.getTime());
        pStmnt.setDate(i,dbdate);
        }
      else pStmnt.setString(i,val);
      }catch(java.text.ParseException e){
        throw new SQLException("setParamStr failed on ["+val+"] as date:"+e);
      } }
    public void setParam(int i, Object ob)throws SQLException{
      if(ob instanceof String)setParamStr(i,(String)ob);
      else if(ob instanceof String[]){
        theStrSeqList=new StrSeqList((String [])ob,i,theStrSeqList);
        }
      else {
        lg.logIt("doQuery.setParam: invalid param "+i);
        setParamStr(i,null);
        } }
    public RowSequence getQueryRows(Env qInfo)
      throws SQLException{
    argCount=1; Object ob=null;
    int maxArgs=qInfo.getInt("ParameterMax",1000);
    checkPstmnt();
    theStrSeqList=null;
    while (argCount<=maxArgs &&
        null!=(ob=qInfo.get("Parameter"+argCount)))   {
    setParam(argCount++,ob);
     }
    qInfo.put("NumberOfParameters",""+(argCount-1));
    qInfo.put("dbQueryString",qString);
    if (givesResultSet){
      theResult=checkPstmnt().executeQuery();
      return new RowSequence(theResult,qInfo);
     }
    else {int N=0;
     if(null==theStrSeqList)N=pStmnt.executeUpdate();
     else {int lim=theStrSeqList.hd.length;
       for(int i=0;i<lim;i++){
         for(StrSeqList ssl=theStrSeqList;ssl!=null;ssl=ssl.tl)
          setParamStr(ssl.loc,(String)(ssl.hd[i]));
```

110

```
        N+=pStmnt.executeUpdate();
      } }
   qInfo.put("NumberOfRowsAffected",""+N);
   return new RowSequence(null,qInfo);
  }
}
}
}
```

APPENDIX I. DBCONNECTIONMANAGER.JAVA

```java
package mythesis.utilityClasses;
import java.util.*;
import java.sql.*;
public class DBConnectionManager extends Cache { //singleton class
static private PropertyGroups pG;
static private Hashtable drivers;
static Logger errLg,adminLg;
static String dbUser=null,dbPwd=null,dbUrl=null; //defaults
static int dbTimeout=100,dbInitSize=2,dbMaxSize=3;
protected DBConnectionManager()throws Exception{
  pG=new PropertyGroups("/DBConnMgr.properties");
  adminLg=new Logger(pG.getProperty("admin.log","DBadmin.log"));
  errLg=new Logger(pG.getProperty("err.log","DBerr.log"));
adminLg.logIt("dbconnectionmanager properties "+pG);
  dbUrl=pG.getProperty("dbUrl",dbUrl); // alter defaults if needed
  dbUser=pG.getProperty("dbUser",dbUser);
  dbPwd=pG.getProperty("dbPwd",dbPwd);
  dbTimeout=topIntProp("dbTimeout",dbTimeout);
  dbMaxSize=topIntProp("dbMaxSize",dbMaxSize);
  dbInitSize=topIntProp("dbInitSize",dbInitSize);
  initDrivers(pG.getProperties("driver"));
  Enumeration pools=pG.propertyKeys();
  while(pools.hasMoreElements()){
    String name=(String)pools.nextElement();
    if(!"driver".equals(name))
      initPool(name,pG.getProperties(name));
    }}
 public void addDriver(String name)throws Exception{
  if(null==name || name.length()==0 || null!=drivers.get(name))return;
  try{
    Driver driver=(Driver)Class.forName(name).newInstance();
    DriverManager.registerDriver(driver);
    drivers.put(name,driver);
    adminLg.logIt("Registered JDBC driver: "+name);
  }catch(Exception ex){
    errLg.logIt("can't register JDBC driver: "+name);
    throw new Exception("can't register JDBC driver: "+name);
  }}
 protected void initDrivers(Properties drivernames)throws Exception{
  drivers=new Hashtable();
  Enumeration names=drivernames.keys();
  while(names.hasMoreElements())
    addDriver((String)names.nextElement());
```

112

```java
    }
    private int topIntProp(String key,int def)throws Exception{
     String S=pG.getProperty(key);
     if(null==S)return def;
     return intKey(key,S);
    }
   private int intKey(String key,String intStr)throws Exception{
    try{return Integer.parseInt(intStr);}
    catch(Exception ex){
      String msg="DBConnectionManager: integer key "+key+"='"+intStr+"'";
      errLg.logIt(msg);
      throw new Exception(msg);
      }}
   private int getInt(Properties p,String key,int def)throws Exception{
    int R=def;
    String S=p.getProperty(key,pG.getProperty(key));
    if(null!=S)return intKey(key,S);
    return R;
   }
   private void initPool(String poolName,Properties props)
    throws Exception{
    String url=props.getProperty("dbUrl",dbUrl);
    if(null==url || url.length()==0){
      String msg="No dbUrl for connection pool "+poolName+" in "+props;
      errLg.logIt(msg);
      throw new Exception(msg);
      }
    String usr=props.getProperty("dbUser",dbUser);
    String pwd=props.getProperty("dbPwd",dbPwd);
    int timeout=getInt(props,"dbTimeout",dbTimeout);
    int maxSize=getInt(props,"dbMaxSize",dbMaxSize);
    int initSize=getInt(props,"dbInitSize",dbInitSize);
    DBConnectionPool cP=
      new DBConnectionPool(url+"--"+usr,url,usr,pwd,timeout,initSize,maxSize,
                  errLg,adminLg);
    put(url,usr,pwd,cP);
    adminLg.logIt("created pool "+poolName+" for "+url+"--"+usr+" "+
            new java.util.Date());
   }
   public DBConnectionPool getConnectionPool(String url,
                  String usr,String pwd)throws Exception{
    if(null==url || url.length()==0)url=dbUrl;
    if(null==url || url.length()==0)
      throw new Exception("no dbUrl for connection pool");
```

113

```java
      DBConnectionPool cP=(DBConnectionPool)get(url,usr,pwd);
      if(null!=cP)adminLg.logIt("retrieved pool for "+url+"--"+usr);
      if(null!=cP)return cP;
      cP=new DBConnectionPool(url+"--"+usr,url,usr,pwd,
                    dbTimeout,dbInitSize,dbMaxSize,
                    errLg,adminLg);
    put(url,usr,pwd,cP);
    adminLg.logIt("created new pool "+url+"--"+usr+" "+new java.util.Date());
    return cP;
    }
  public boolean freeItem(Object ob){
    // called by freeSpace;
    if(!(ob instanceof DBConnectionPool)){
      errLg.logIt("non-connectionpool in DBConnectionManager cache!"+ob);
      return false;
      }
    DBConnectionPool cP=(DBConnectionPool)ob;
    cP.close();
    return true;
    }
  private static Cache instance=null;
  private static int clients=0;
  protected void init(){super.init();}
  public static synchronized Cache getInstance(){
    try{
      if(null==instance)instance=new DBConnectionManager();
      clients++;
      adminLg.logIt("added new ConnectionManager instance");
      return instance;
    }catch(Exception ex){ex.printStackTrace(); return null;}
  }
  public static synchronized int freeInstance(){
    if(null==instance)return 0;
    clients--;
adminLg.logIt("freed instance of connectionmanager, leaving "+clients);
    if(clients==0)
      try{close();
      adminLg.logIt("closed connection manager");
      }catch(Exception ex){errLg.logIt("freeInstance ",ex);}
    return clients;
  }
  public static synchronized boolean close()throws Exception{
    clients=0;
    while(0==instance.freeSpace(10000));
```

114

```java
    instance=null;
    Enumeration enum=drivers.keys();
    while(enum.hasMoreElements()){
      String key=(String)enum.nextElement();
      try{
        DriverManager.deregisterDriver((Driver)drivers.get(key));
        adminLg.logIt("Deregistered driver "+key);
      }catch(Exception ex){
        String msg="failed to deregister driver "+key;
        errLg.logIt(msg);
        throw new Exception(msg); // or return false, or just skip it
      }}
    return true;
  }
}
```

APPENDIX J. DBCONNECTIONPOOL.JAVA

```java
package mythesis.utilityClasses;
import java.util.Date;
import java.sql.Connection;
import java.sql.SQLException;
import java.sql.DriverManager;
public class DBConnectionPool{
  int maxSize, initSize, inUse, timeout;
  String poolName, dbUrl, dbUser, dbPwd;
  Queue Q;
  Logger errLg;
  Logger adminLg;
public DBConnectionPool
  (String nm,String url,String usr,String pwd,
        int timeout,int initSize,int maxSize,
      Logger er,Logger adm) throws Exception{
  // save all parameters values in its instance variables
  poolName=nm;
  dbUrl=url;
  dbUser=usr;
  dbPwd=pwd;
  if(null==dbUser)dbUser="";
  if(null==dbPwd)dbPwd="";
  this.maxSize=maxSize; this.initSize=initSize; this.timeout=timeout;
  errLg=er; adminLg=adm;
  Q=new Queue(); inUse=0;
  adminLg.logIt("DBConnectionPool init for "+poolName);
  preload(initSize);}
public synchronized Connection popConnection(){
  while(!Q.isEmpty()){
   try{
    Connection con=(Connection)Q.next();
    if(!con.isClosed())return con; // might be closed by dbase
    adminLg.logIt("REJECT: connection in "+poolName+" was closed");
    }catch(Exception ex){
     errLg.logIt("DBConnectionPool.popConnection: ",ex);
    }}
  return null;}
public synchronized Connection getConnection(){
  Connection con=popConnection();
  if(null==con)
   if(maxSize<0 || inUse<maxSize)
     con=newConnection();
  if(null!=con)inUse++;
```

116

```java
    return con;}
  public synchronized void close(){
    Connection con;
    while(!Q.isEmpty()){
     if(null==(con=popConnection()))continue;
     try{
      con.close();
      adminLg.logIt("closed connection in pool "+poolName);
     }catch(SQLException ex){errLg.logIt("err in closing "+poolName,ex);}}
    if(inUse>0)
     errLg.logIt("close "+poolName+" with "+inUse+" still connected");
    else adminLg.logIt("closed "+poolName);}
  public void preload(int N)throws Exception {
    for(int i=0;i<N;i++){
     Connection con=newConnection();
     if(null==con)throw new Exception("connection failure in "+dbUrl);
     Q.append(con);
     }}
  public synchronized void freeConnection(Connection con){
    Q.append(con);
    inUse--;
    notifyAll(); }// wake up any waiting in getConnection(timeout);
  private Connection newConnection(){
    adminLg.logIt("newConnection    for    pool    "+poolName+";    "+dbUser+";
"+dbPwd);
      try{
       if(dbUser.length()==0)
         return DriverManager.getConnection(dbUrl);
       else return DriverManager.getConnection(dbUrl,dbUser,dbPwd);
      }catch(SQLException ex){
       errLg.logIt("no newConnection for "+dbUrl,ex);
       return null;
      }}
  public Connection getConnection(int timeout){
    long waitUntil=new Date().getTime()+timeout;
    Connection con;
    while(null==(con=getConnection()) && waitUntil > new Date().getTime()){
     try{wait(timeout);}catch(InterruptedException ex){}
     }
    return con;}
  public Connection getConnWait(){
   return getConnection(timeout);
  }
  }
```

APPENDIX K. HTMLWRAPPER.JAVA

```java
package mythesis.utilityClasses;
import java.io.*;
import java.util.Enumeration;
import org.apache.ecs.html.*; // used only for wrapPageECS, wrapTablePageECS

public class HtmlWrapper {
 PrintWriter out=null;
 Logger lg=null;
 public HtmlWrapper (PrintWriter o){out=o; lg=new Logger();}
 public void wrapBodyPage(String title,String body) throws IOException {
   wrapHeader(title);
   openBody();
   wrapHeading(title,1);
   out.println(body);
   closeBody();
   out.close();
 }
 public void wrapTablePage(String title,String[][]tab)
              throws IOException{
   wrapHeader(title);
   openBody();
   wrapHeading(title,1);
   wrapTable(tab);
   closeBody();
   out.close();
 }
 public void wrapTablePage(String title,String[][]tab,String xtra)
              throws IOException{
   wrapHeader(title,xtra);
   openBody();
   wrapHeading(title,1);
   wrapTable(tab);
   closeBody();
   out.close();
 }
 public void wrapEnvResultPage(Env resEnv) throws IOException{
  String numRows=resEnv.getStr("NumberOfRowsAffected");
  if(numRows!=null) wrapPage("Rows Affected:",numRows);
  else wrapTablePage("Query Response",
              (String[][])resEnv.get("ResultTable"));
 }
 public void wrapRowsPage(RowSequence dbRows,Env rqEnv){
  String fName=rqEnv.getStr("templateFile");
```

118

```java
    String fValue=null;
    try{
     fValue=MiscFile.fileSubstByTag(fName,dbRows);
     }catch(Exception ex){
      lg.logIt(""+ex);ex.printStackTrace(out);return;
      }
     out.println(fValue);
     out.close();
    }
 public void wrapEnvPage(Env E){
  String fName=E.getStr("templateFile");
  String fValue=null;
  try{
   fValue=MiscFile.fileSubstByTag(fName,E);
   if(null==fValue)throw new Exception("no file from "+fName);
   }catch(Exception ex){
    ex.printStackTrace(out);
    lg.logIt("HtmlWrapper.wrapEnvPage: ",ex);return;
   }
  out.println(fValue);
  out.close();
  }
  public void wrapHeader(String title)throws IOException{
    out.println("<HTML><HEAD><TITLE> "+title+" </TITLE></HEAD>");
  }
 public void wrapHeader(String title,String xtra)throws IOException{
    out.println("<HTML><HEAD><TITLE> "+title+" </TITLE>\n"+
           xtra+"</HEAD>");
  }
  public void wrapHeading(String H,int level)throws IOException{
   out.println("<H"+level+">"+H+"</H"+level+">");
  }
  public void openBody()throws IOException{
   out.println("<BODY>");
  }
  public void closeBody()throws IOException{
   out.println("</BODY></HTML>");
  }
  public void wrapTable(String [][]tab)throws IOException{
   if(tab==null)return;
   openTable();
   wrapTableHeaders(tab[0]);
   for(int i=1;i<tab.length;i++)wrapTableRow(tab[i]);
   closeTable();
```

119

```java
}
public void wrapTableHeaders(String [] H)throws IOException{
  out.println("<TH>"+Misc.stringArrayJoin(H,"</TH><TH>")+"</TH>");
}
public void wrapTableRow(String [] R)throws IOException{
 out.println("<TR>");
 out.println("<TD>"+Misc.stringArrayJoin(R,"</TD><TD>")+"</TD>");
 out.println("</TR>");
}
public void openTable()throws IOException{
 out.println("<TABLE>");
}
public void closeTable()throws IOException{
 out.println("</TABLE>");
}
public void wrapPage(String title,String body) throws IOException {
 wrapBodyPage(title,body);
 }
public void wrapPageECS(String title,String body) throws IOException {
 Html thePage=new Html()
        .addElement(new Head()
           .addElement(new Title(title)))
        .addElement(new Body()
           .addElement(new H1(title))
           .addElement(body));
 out.println(thePage.toString());
 out.close();
}
 public void wrapTablePageECS(String title,String[][]tab)
              throws IOException{
 Table theTable=new Table();
 TR heads=new TR();
 for(int i=0;i<tab[0].length;i++)
  heads.addElement(new TH().addElement(tab[0][i]));
 theTable.addElement(heads);
 for(int j=1;j<tab.length;j++){
  TR theRow=new TR();
  for(int i=0;i<tab[j].length;i++)
    theRow.addElement(new TD().addElement(tab[j][i]));
  theTable.addElement(theRow);
  }
 Html thePage=
  new Html()
    .addElement(new Head()
```

```java
            .addElement(new Title(title)))
          .addElement(new Body()
            .addElement(new H1(title))
            .addElement(theTable));
      out.println(thePage.toString());
      out.close();
      }
    public void wrapRowsPageLines(RowSequence dbRows,Env rqEnv){
      RowSubst rSub=new RowSubst(rqEnv.getStr("templateFile"),dbRows,out);
      rSub.interpret();
        out.close();
      }
    public void wrapEnvPageLines(Env E){
     String fName=E.getStr("templateFile");
     String fValue=MiscFile.substLines(fName,E);
     out.println(fValue);
        out.close();
      }
    }
```

APPENDIX L. MISCDB.JAVA

```java
    package mythesis.utilityClasses;
    import java.sql.*;
    import java.util.Hashtable;
    import java.util.Vector;
    import java.io.*;

    public class MiscDB {
    public static Hashtable resultSetAsHashTable(ResultSet R)
        throws SQLException {
      Hashtable H=new Hashtable();
      if(R==null)return H;
      String[] fieldNames=resultSetLabels(R);
      return resultSetAsHashTable(fieldNames,R,H);
    }
    public static Hashtable resultSetAsHashTable(String [] cols,ResultSet R,
                           Hashtable H)
        throws SQLException {
      if(R==null)return H;
      for (int i=1; i<=cols.length;i++)
       H.put(cols[i-1],R.getString(i));
      return H;
```

```java
      }
    // get column names as a string array
    public static String [] resultSetLabels(ResultSet R)
          throws SQLException{
     ResultSetMetaData rsmd = R.getMetaData();
     String S []=new String[rsmd.getColumnCount()];
     for(int i=0;i<S.length;i++)S[i]=rsmd.getColumnLabel(i+1);
     return S;
    }
    // get the number of columns in the result set.
    public static int resultSetColumnCount(ResultSet R)
          throws SQLException{
     ResultSetMetaData rsmd = R.getMetaData();
     return rsmd.getColumnCount();
    }
    // get the data types of the columns as string array
    public static String [] resultSetTypes(ResultSet R)
          throws SQLException{
     ResultSetMetaData rsmd = R.getMetaData();
     String S []=new String[rsmd.getColumnCount()];
     for(int i=0;i<S.length;i++)S[i]=rsmd.getColumnTypeName(i+1);
     return S;
    }
    // get the result set values as row
    public static String [] resultRowValues(ResultSet R)
          throws SQLException{
     String S[]=new String[resultSetColumnCount(R)];
     for(int   i=0;i<S.length;i++)S[i]=R.getString(i+1);// java   array   is   0-based   ,
resultset is 1-based
     return S;
    }
    public static String[][] vectorToStringMatrix(Vector V){
      // V is actually a vector of string-arrays.
      if(V==null || V.size()==0)return null;
      String [][]R=new String[V.size()][];
      for(int i=0;i<R.length;i++)R[i]=(String[])(V.elementAt(i));
      return R;
    }
    public static String[][] resultRowsToStringMatrix(ResultSet R){
      try{
        Vector V=new Vector(); // result
        V.addElement(resultSetLabels(R)); // table header line
        while(R.next())V.addElement(resultRowValues(R));
        R.close();
```

122

```java
      return vectorToStringMatrix(V);
     }catch(SQLException E){E.printStackTrace();return null;}
   }
public static String  createTable(Statement stmnt, String name,
                 String[]fldnames,String[]fldtypes,boolean doReset){
  String result="";
  try{
   int N=fldnames.length;
   String createStr="CREATE TABLE "+name+" (";
   if(N>0)createStr+=fldnames[0]+" "+fldtypes[0];
   for(int i=1;i<N;i++)createStr+=", "+fldnames[i]+" "+fldtypes[i];
   createStr+=")";
   result+="createStr="+createStr+"\n";
   try{stmnt.execute(createStr);
   }catch(SQLException e){
     StringWriter sw=new StringWriter();
     e.printStackTrace(new PrintWriter(sw));
     result+=sw.toString();
     }
    // this table may already exist;
   if(doReset){
    String delStr="DELETE * FROM "+name;
    stmnt.execute(delStr);
    }
  }catch(SQLException e){
   StringWriter sw=new StringWriter();
    e.printStackTrace(new PrintWriter(sw));
   result+=sw.toString();
    }
  return result;
 }
public static PreparedStatement
   createInserter(String name,String[]fldnames,Connection conn)
    throws SQLException {
   int N=fldnames.length;
   String insertStr="INSERT INTO "+name+" VALUES (";
   if(N>0)insertStr+="?";
   for(int i=1;i<N;i++)insertStr+=", ?";
   insertStr+=")";
   PreparedStatement pStmnt=conn.prepareStatement(insertStr);
   return pStmnt;
 }
}
```

APPENDIX M. MISC.JAVA

```java
package mythesis.utilityClasses;
import java.util.Hashtable;
import java.util.Enumeration;
import java.util.Vector;
import java.sql.ResultSet; // for conversions

public class Misc {
public static Hashtable hashDefs(String [] names, String [] values){
  Hashtable H=new Hashtable();
  if(names.length>values.length)return H;
  for(int i=0;i<names.length;i++)H.put(names[i],values[i]);
  return H;
}
public static String [] lookupHash(String [] names, Hashtable H){
  if(H==null)return new String[0];
  String [] R= new String[names.length];
  for(int i=0;i<names.length;i++)R[i]=(String)H.get(names[i]);
  return R;
}
public static String htmlEscape(String S){
  if(null==S)return S;
  int N=S.length();
  StringBuffer sb=new StringBuffer(N);
  for(int i=0;i<N;i++){
    char c=S.charAt(i);
    if(c=='&')sb.append("&amp;");
    else if(c=='"')sb.append("&quot;");
    else if(c=='<')sb.append("&lt;");
    else if(c=='>')sb.append("&gt;");
    else if(c=='\'')sb.append("&#39;");
    else sb.append(c);
    }
  return sb.toString();
}
public static String stringArrayJoin(String [] A, String S){
  if(A==null || A.length==0)return "";
  StringBuffer sb=new StringBuffer();
  sb.append(A[0]);
  for(int i=1;i<A.length;i++){sb.append(S);sb.append(A[i]);}
  return sb.toString();
}
public static String [] stringSplit(String S,char delim){
  Vector V=new Vector();
```

```java
    StringSplitter SS=new StringSplitter(S,delim);
    while(SS.hasMoreTokens())V.addElement(SS.nextToken());
    return vectorToStringArray(V);
  }
 public static String [] stringSplit(String S,String delim){
   Vector V=new Vector();
   StringSplitter SS=new StringSplitter(S,delim);
   while(SS.hasMoreTokens())V.addElement(SS.nextToken());
   return vectorToStringArray(V);
 }
 public static String [] stringSplit(String S){ // delim==S[0]
  if(S==null || S.length()==0)return new String[0];
  char delim=S.charAt(0);
  Vector V=new Vector();
  StringSplitter SS=new StringSplitter(S,delim,1);
  while(SS.hasMoreTokens())V.addElement(SS.nextToken());
  return vectorToStringArray(V);
 }
 public static Hashtable splitDelimHash(String S){//delim=S[0]
  Hashtable H=new Hashtable(1);
  if(S==null || S.length()==0)return H;
  char delim=S.charAt(0);
  StringSplitter SS=new StringSplitter(S,delim,1);
  while(SS.hasMoreTokens()){
    String k=SS.nextToken();
    if(SS.hasMoreTokens())H.put(k,evalQuotedChars(SS.nextToken()));
    }
  return H;
 }
 public static String stringDelimSubst(String S,String d,Env defs){
  // S contains keys, beginning and ending with copies of delim;
  // result is to be that of replacing these with their values
  String [] A =stringSplit(S,d);
  for(int i=1;i<A.length;i+=2)A[i]=defs.getStr(A[i]);
  return stringArrayJoin(A,"");
 }
 public static String stringDelimSubst(String S,String d,Hashtable defs){
  // S contains keys, beginning and ending with copies of delim;
  // result is to be that of replacing these with their values
  String [] A =stringSplit(S,d);
  for(int i=1;i<A.length;i+=2)A[i]=(String)defs.get(A[i]);
  return stringArrayJoin(A,"");
 }
 public static String substLineByTags(String S,Env env)
```

125

```java
      throws ParseSubstException{
       if(null==S)return "";
       StringBuffer sBuff=new StringBuffer(S);
       ParseSubst pS=new ParseSubst(sBuff);
       return pS.toString(env);
    }


   public static String substLine(String S,Env env){
      if(S==null || !S.startsWith("$$SUBST:"))return S;
      int dPos=8; int dEnd=S.indexOf(':',9);
      if(dEnd<=dPos)return S; //invalid input, no matching ":"
      String d=S.substring(dPos,dEnd);
      return stringDelimSubst(S.substring(dEnd,S.length()),d,env);
      }
   public static String substFile(String fName, String fDelim, String defs){
     Hashtable dict=splitDelimHash(defs);
     if(dict==null)return("no definitions for "+fName+" in "+defs);
     return stringDelimSubst(MiscFile.fileToString(fName),fDelim,dict);
     }
   public static String indent(int Level){
     String S="";while(0<Level--)S+="  ";return S;}


   public static int getInt(String S,int dval){
     if(S==null)return dval;
     try{int N=Integer.parseInt(S);return N;}
     catch(Exception e){return dval;}
   }
   public static String getStr(String S,String dval){
     if(S==null)return dval;
     return S;
   }
   public static String evalQuotedChars(String S){
     String R="";
     for(int i=0;i<S.length();i++){
      char c=S.charAt(i);
      if(c!='\\')R+=""+c;
      else {i++;R+=""+S.charAt(i);}
      }
     return R;
   }
   public static String quoteSpecialChars(String S,String specials){
     String R=""; // should use stringbuffer for efficiency?
     for(int i=0;i<S.length();i++){
      char c=S.charAt(i);
```

126

```java
      if(specials.indexOf(c)>=0)R+="\\"+c;
      else R+=""+c;
      }
    return R;

  }
  public static String hashAttribString(Hashtable H){
   // returns the attribute string
   Enumeration KK=H.keys();
   String S=""; String specialChars="\\\"";
   while(KK.hasMoreElements()){
     String k=(String)KK.nextElement();
     String v=(String)H.get(k);
     S+=" "+k+"=\""+quoteSpecialChars(v,specialChars)+"\"";}
   return S;
  }
  public static Hashtable attribStringHash(String S){
   // interprets the attribute string.
   // but closing "quote" is required, and the string must be
   // _delimited_ by blanks; no error checking yet.
   Hashtable H=new Hashtable(); int loc=0; int lim=S.length();
   while(loc<lim && ' '==S.charAt(loc))loc++;
   while(loc<lim){
     int eqLoc=S.indexOf("=",loc);
     if(eqLoc<0)return H;
     String k=S.substring(loc,eqLoc);
     char q=S.charAt(eqLoc+1);
     int endLoc=eqLoc+2; char c;
     while(endLoc<lim && (c=S.charAt(endLoc))!=q)
       if(c=='\\')endLoc+=2; else endLoc++;
     if(endLoc>lim)return H; // no closing quote
     String v=S.substring(eqLoc+2,endLoc);
     H.put(k,evalQuotedChars(v));
     loc=endLoc+2;
     while((loc<lim) && ' '==S.charAt(loc))loc++;
     }
   return H;
  }
  public static String[] vectorToStringArray(Vector V){
  String [] S = new String[V.size()];
  for(int i=0;i<S.length;i++)S[i]=(String)V.elementAt(i);
  return S;
  }
  }
```

APPENDIX N. MISCDATE.JAVA

```java
package mythesis.utilityClasses;
import java.util.GregorianCalendar;
import java.util.Calendar;

public class MiscDate {
public static String todaysDate(){
  Calendar C=new GregorianCalendar();
  return C.get(Calendar.MONTH)+"/"+
      C.get(Calendar.DAY_OF_MONTH)+"/"+
      C.get(Calendar.YEAR) + "@"+
      C.get(Calendar.HOUR) + ":"+
      C.get(Calendar.MINUTE) + ":"+
      C.get(Calendar.SECOND) + ":"+
      C.get(Calendar.MILLISECOND)+
      (Calendar.AM==C.get(Calendar.AM_PM)?"AM":"PM");
}
}
```

APPENDIX O. MISCFILE.JAVA

```java
package mythesis.utilityClasses;
import java.io.*;
import java.util.Hashtable;
import java.util.Enumeration;

public class MiscFile {
public static StringBuffer fileToStringBuffer(String fName) {
  StringBuffer sBuff=new StringBuffer();
  InputStreamReader inStr=null;
  try{
   inStr=new InputStreamReader(new FileInputStream(fName));
   char[] cBuff=new char[4096];
   int charsRead;
   while(-1 != (charsRead=inStr.read(cBuff)))
     sBuff.append(cBuff,0,charsRead);
  }catch(Exception ex){ex.printStackTrace();return null;}
  finally{
   try{inStr.close();}catch(Exception ex){}
   }
  return sBuff;
}
public static String fileSubstByTag(String fName,Env env)
   throws Exception{
  StringBuffer sBuff=fileToStringBuffer(fName);
```

```java
      if(sBuff==null)throw new Exception("no file for "+fName);
      ParseSubst pS=new ParseSubst(sBuff);
      return pS.toString(env);
   }
  public static String fileSubstByTag(String fName,RowSequence rows)
     throws ParseSubstException{
     StringBuffer sBuff=fileToStringBuffer(fName);
     if(sBuff==null)throw new ParseSubstException("no file for "+fName);
     ParseSubst pS=new ParseSubst(sBuff);
     return pS.toString(rows);
   }
  public static String fileToString(String fName) {
     String content="";BufferedReader brin=null;
     try{
       brin=new BufferedReader(new FileReader(fName));
       String nextLine;
       while(null!=(nextLine=brin.readLine()))content+=nextLine+"\n";
       brin.close();
       }catch(IOException e)
         {try{brin.close();}catch(IOException ex){}}
     return content;
     }
 public static String substLine(String L,Hashtable dict){
   // a preliminary version of the substition within DBFileSubst
   String theCommandPrefix="$$SUBST";
   if(L==null)return "";
   if(!L.startsWith(theCommandPrefix))return L;
   try{
     int cmdBegin=theCommandPrefix.length();
     int cmdEnd=L.indexOf(':');
     int delimEnd=L.indexOf(':',1+cmdEnd);
     String cmd=L.substring(2,cmdEnd);
     String delim=L.substring(1+cmdEnd,delimEnd);
     L=L.substring(1+delimEnd,L.length());
     return Misc.stringDelimSubst(L,delim,dict);
     }catch(Exception E){}
   return L;
 }
 public static String substLines(String fName,String defs) {
   return substLines(fName,Misc.splitDelimHash(defs));
 }
 public static String substLines(String fName,Hashtable dict) {
   String content="";BufferedReader brin=null;
   try{
```

```
      brin=new BufferedReader(new FileReader(fName));
      String nextLine;
      while(null!=(nextLine=brin.readLine()))
        content+=substLine(nextLine+"\n",dict);
      brin.close();
      }catch(IOException e)
        {try{brin.close();}catch(IOException ex){}}
    return content;
  }
  public static BufferedReader getBufferedReader(String fName){
   try{
    FileReader fR=new FileReader(fName);
    if(fR==null)return null;
    return new BufferedReader(fR);
   }catch(Exception ex)
     {ex.printStackTrace();
      return null;}
  }
  }
```

APPENDIX P. MYNALEX.JAVA

```
      package mythesis.utilityClasses;
      import java.util.Hashtable;
      import java.util.Stack;

      public class MyNaLex {
       public static final int noToken = -1;
       public static final int textToken = 0;
       public static final int mynaToken = 1;
       public static final int endMynaToken = 2;
       public static final int endAllToken = 3;
       public static String defaultDelim = "|";
       private StringBuffer sBuff=null;
       private String theString=null; // theString is a reference to sBuff.
       private int sBuffLength;
       private int curLoc;     // curLoc will shift through sBuff as we read.
       private int tokenType; // these four properties _are_ current token.
       private int tokenStart;
       private int tokenEnd;
       private Hashtable tokenProps=null;
       public static String[]tokTypeNames= //noToken==-1
         new String[]{"noToken","textToken",
                 "mynaToken","endMynaToken","endAllToken"};
       private int pushbackTokenType; // these four are a token which has
```

```java
private int pushbackTokenStart; // been recognized but not yet yielded
private int pushbackTokenEnd;
private Hashtable pushbackTokenProps;
private String currentDelimiter; // null outside of range.
private Stack delimStack; // nested ranges, multiple delimiters.
Logger lg; private static boolean logOn=false;
public MyNaLex(StringBuffer sB){
  lg=new Logger();
  sBuff=sB;
  if(sBuff==null)
    {lg.logIt("null string buff for MyNaLex"); sBuff=new StringBuffer();}
  theString=sBuff.toString();
  sBuffLength=theString.length();
  curLoc=0;
  tokenStart=0; tokenEnd=0; tokenType=noToken; tokenProps=null;
  currentDelimiter=null; delimStack=null;
  pushbackTokenType=noToken;
}
public int getTokenStart(){return tokenStart;}
public int getTokenEnd(){return tokenEnd;}
public int getTokenType(){return tokenType;}
public Hashtable getTokenProps(){return tokenProps;}
public int setToken(int lo,int hi,int tType,Hashtable tProps){
  tokenStart=lo; tokenEnd=hi; tokenType=tType; tokenProps=tProps;
  if(tokenType==endMynaToken){
    if(delimStack==null || delimStack.empty())
      currentDelimiter=null;
    else currentDelimiter=(String)delimStack.pop();
  }
  else if(tokenType==mynaToken){
    if(currentDelimiter!=null){ // this only happens in nested <myThesis:
      if(delimStack==null)delimStack=new Stack();
      delimStack.push(currentDelimiter);
    }
    String delim;
    if(tProps!=null && null!=(delim=(String)tProps.get("delim")))
      currentDelimiter=delim;
    else currentDelimiter=defaultDelim;
  }
  return tokenType;
}
public String getTokenString(){
  if(tokenStart<tokenEnd)
    return theString.substring(tokenStart,tokenEnd);
```

131

```
        if(tokenStart==tokenEnd)return "";
        lg.logIt("tokenStart>tokenEnd,"+tokenStart+">"+tokenEnd);
        return "";
      }
    public int getToken(){tokenProps=null;
      if(logOn)lg.logIt("getToken called with curLoc="+curLoc+
          ", prevTokType="+tokTypeNames[1+tokenType]+
          ", prevTokVal:\n"+
          (tokenType<0?"":getTokenString()+
          ",context="+context()));
      if(pullforthToken())return tokenType; // previously matched
      if(curLoc>=sBuffLength)return tokenType=endAllToken; // nothing there

      if(curLoc==0){
        int firstLoc=theString.indexOf("<myThesis:");
        if(firstLoc<0) return setToken(0,curLoc=sBuffLength,textToken,null);
        }
      int oldCurLoc=curLoc; // location at start of getToken1 call
      while(curLoc<sBuffLength){
        char c=sBuff.charAt(curLoc); // another manual optimization;
        if(c!='<' &&     // if these tests fail, so would matching below.
          (currentDelimiter==null || c!= currentDelimiter.charAt(0)) )
          {curLoc++; continue;} // end manual optimization section
        int startMatch=curLoc; // location as we start trying to match.
        if(getMynaToken())
          return pushbackToken(oldCurLoc,startMatch,textToken,null);
        if(getEndMynaToken())
          return pushbackToken(oldCurLoc,startMatch,textToken,null);
        if(null!=currentDelimiter && getStr(currentDelimiter))
          return setToken(oldCurLoc,startMatch,textToken,null);
        curLoc++;
        }
      return setToken(oldCurLoc,sBuffLength,textToken,null);
    }
  public boolean getLetter(){
    if(curLoc>=sBuffLength) return false;
    char c=sBuff.charAt(curLoc);
    if( ('a' <= c && c <= 'z')
       || ('A' <= c && c <= 'Z')
       || (c=='_' || c=='#') )
      {curLoc++; return true;}
    return false;
  }
  public boolean getChar(char c){
```

132

```java
   if(curLoc>=sBuffLength) return false;
   if(c!=sBuff.charAt(curLoc))return false;
   curLoc++;
   return true;
 }
 public boolean getCharRange(char lo, char hi){
   if(curLoc>=sBuffLength) return false;
   char c=sBuff.charAt(curLoc);
   if(c<lo || c>hi)return false;
   curLoc++;
   return true;
 }
 public boolean getWhiteSpace(){
   if(curLoc>=sBuffLength) return false;
   char c=sBuff.charAt(curLoc);
   if(c>' ')return false;
   while((++curLoc<sBuffLength) && (' '>sBuff.charAt(curLoc)));
   return true;
 }
 public boolean getStr(String S){
   if(theString==null){lg.logIt("null string in getStr"); return false;}
   if(S==null){lg.logIt("null match for getStr"); return false;}
   if(!theString.startsWith(S,curLoc))return false;
   curLoc+=S.length();
   return true;
 }
 public boolean getId(){
   int start=curLoc;
   if(!getLetter())return false;
   while(getLetter()); // curLoc now points past the end of the id.
   return true;
 }
 public boolean getQStr(){
   int start=curLoc;
   if(curLoc>=sBuffLength)return false;
   char qChar= sBuff.charAt(curLoc++);
   while (curLoc<sBuffLength && qChar!=sBuff.charAt(curLoc))curLoc++;
   if(curLoc>=sBuffLength){curLoc=start; return false;}
   curLoc++; // step past the matching quote character;
   return true;
 }
 public boolean getDef(){
   int startId=curLoc;
   if(!getId())return false;
```

```
        int endId=curLoc; // endId points to the '=', or should
        getWhiteSpace();
        if(!getChar('=')){curLoc=startId; return false;}
        getWhiteSpace();
        if(!getQStr()){curLoc=startId; return false;}
        if(tokenProps==null)tokenProps=new Hashtable(1);
        tokenProps.put(theString.substring(startId,endId),
              theString.substring(endId+2,curLoc-1));
        return true;
      }
     public boolean getMynaToken(){
     if(logOn)lg.logIt("gMT: "+context());
      int startTok=curLoc;
      if(!getStr("<myThesis:")){ return false;}
      int startId=curLoc;
      if(!getId()){curLoc=startTok; return false;}
      int endId=curLoc;
      if(getWhiteSpace())
        while(getDef())
          while(getWhiteSpace());
      if(!getChar('>')){curLoc=startTok;  lg.logIt("gMT  failed  1"+context());  return
false;}
        setToken(startTok+1,endId,mynaToken,tokenProps); // skip "<"
      if(logOn)lg.logIt("gMTend: tokenStart="+tokenStart+", tokenEnd="+tokenEnd+
            ",currentDelim="+currentDelimiter+
            ",tokenType="+tokTypeNames[1+tokenType]+"context="+context());
      return true;
      }
     public boolean getEndMynaToken(){
      int startTok=curLoc;
      if(!getStr("</myThesis:")) return false;
      int startId=curLoc;
      if(!getId()){ curLoc=startTok; return false;}
      int endId=curLoc;
      while(getChar(' '));
      if(!getChar('>')){ curLoc=startTok; return false;}
      setToken(2+startTok, endId,endMynaToken,tokenProps); // skip "</"
      return true;
      }
     public String context(){ // used in error messages.
      if(curLoc>=sBuffLength)
        return "end of Buffer:\n  ["+getStr(sBuffLength-20,sBuffLength);
      int numLines=0; int nl=0;
      for(int i=0;i<curLoc;i++)
```

134

```java
        if(theString.charAt(i)=='\n'){numLines++;nl=i;}
      String msg= "MyNaTok Context:\nlline "+numLines+", char "+(curLoc-nl)
                    +"\nl"+getStr(nl,curLoc);
      for(int i=curLoc;i<sBuffLength && theString.charAt(i)!='\n';i++)
        msg+=theString.charAt(i);
      msg+="\nl";
      int lastC=curLoc-1;
      for(int i=nl;i<lastC;i++)msg+=".";
      msg+="*";
      return msg;
    }
  public String getStr(int lo, int hi){
    if(lo>=hi)return "";
    if(lo>=sBuffLength || hi<0)return "";
    if(lo<0)lo=0;if(hi>sBuffLength)hi=sBuffLength;
    return theString.substring(lo,hi);
  }
    public void pushbackToken(){
      pushbackTokenStart=tokenStart;
      pushbackTokenType=tokenType;
      pushbackTokenEnd=tokenEnd;
      pushbackTokenProps=tokenProps;
      tokenProps=null;
    }
    public int pushbackToken(int lo,int hi,int tt,Hashtable tp){
      pushbackToken();
      return setToken(lo,hi,tt,tp);
    }
    public boolean pullforthToken(){
     if(pushbackTokenType==noToken)return false;
     setToken(pushbackTokenStart,pushbackTokenEnd,
          pushbackTokenType,pushbackTokenProps);
     pushbackTokenType=noToken; // mark as unavailable.
     return true;
    }
}
```

APPENDIX Q. QUEUE.JAVA

```java
package mythesis.utilityClasses;
import java.util.*;

public class Queue extends Vector {
public boolean isEmpty(){return size()<=0;}
public void append(Object ob){addElement(ob);}
public Object next() throws Exception{
  if(isEmpty())throw new Exception("no Queue.next() on empty queue");
  Object ob=firstElement();
  removeElementAt(0);
  return ob;
 }
}
```

APPENDIX R. STRINGSPLITTER.JAVA

```java
package mythesis.utilityClasses;
public class StringSplitter{
//       takes only one delim; this may be a character or a string.
  String theString; char theDelim; int thePos;
  String theDelimStr=null; int theDelimLength;
  // nextToken is the token beginning at thePos
  public StringSplitter(String S,char d,int p){
    theString=S; theDelim=d; thePos=p; theDelimLength=1;
    if(thePos>=theString.length())thePos=-1;
    }
  public StringSplitter(String S,char d){this(S,d,0);}
  public StringSplitter(String S,String d,int p){
    theString=S; theDelimStr=d; thePos=p;
    theDelimLength=d.length();
    if(thePos>=theString.length())thePos=-1;
    }
  public StringSplitter(String S,String d){this(S,d,0);}
  public boolean hasMoreTokens(){return thePos>=0;}
  public String nextToken(){
    if(thePos<0) return null;
    int nextPos;
    if(theDelimStr==null) nextPos=theString.indexOf(theDelim,thePos);
    else nextPos=theString.indexOf(theDelimStr,thePos);
    String R;
    if(nextPos>=0){
      R=theString.substring(thePos,nextPos);
      thePos=nextPos+theDelimLength;
      } else {
```

```
            R=theString.substring(thePos);
            thePos=nextPos;
            }   return R;   }   }                .
```

APPENDIX S. PARSESUBSTEXCEPTION.JAVA

```
        package mythesis.utilityClasses;
        public class ParseSubstException extends Exception{
         public ParseSubstException(){super();}
         public ParseSubstException(String S){super(S);}
        }
```

APPENDIX T. PARSESUBST.JAVA

```
        package mythesis.utilityClasses;
        import java.io.*;
        import java.util.*;
        import java.sql.SQLException;
        public class ParseSubst {
          StringBuffer theBuff; String theString;
          MyNaLex lex=null;
          ParseTree theTree=null;
          Env theEnv=null; RowSequence theRows=null;
          boolean substFailure;
          StringBuffer outBuff;
          Logger lg;
        public String toString(Env env) throws ParseSubstException{
          theEnv=env;
          outBuff=new StringBuffer();
          toStringBuffer(theTree);
          return outBuff.toString();
        }
        public String toString(RowSequence rows) throws ParseSubstException{
          theRows=rows;
          if(null==rows)throw  new   ParseSubstException("empty   RowSequence  in
toString");
         theEnv=theRows.getRow();
         outBuff=new StringBuffer();
         toStringBuffer(theTree);
         return outBuff.toString();
        }
        public String getText(ParseTree T){
         int low=T.getLow(); int high=T.getHigh();
         if(low>=high)return "";
```

```java
    return theString.substring(low,high);
  }
  public String getVal(ParseTree T){
    int low=T.getLow(); int high=T.getHigh();
    if(low>=high)return "";
    return theEnv.getStr(theString.substring(low,high));
  }
  String treeReport(ParseTree T){
    if(T==null)return "<NULLTREE>";
    String tag=T.getTagName();
    String S="<"+tag+">";
    for(int i=0;i<T.numChildren();i++)
      S+=treeReport(T.child(i));
    if("TEXT".equals(tag))S+=getText(T);
    S+="</"+tag+">";
    return S;
  }
  String treeReport(){return treeReport(theTree);}
  public void toStringBuffer(ParseTree T) throws ParseSubstException{
    if(T==null)return;
    int N=T.numChildren();
    String tag=T.getTagName();
    if(tag.equals("ROOT"))
      for(int i=0;i<N;i++)
        toStringBuffer(T.child(i));
    else if(tag.equals("TEXT"))
      outText(T);
    else if(tag.startsWith("myThesis:"))
      substStringBuffer(tag,T);
    else throw new ParseSubstException("expected ROOT or 'myna:', found "+tag);
  }
  public void newQuery(ParseTree T) throws ParseSubstException{
    Hashtable H=T.getProps();
    if(H==null)return;
    String theOp=(String)H.get("dbOperation");
    if(theOp==null)return;
    theEnv.addHashtable(H);
    DBHandler theDBHandler=(DBHandler)theEnv.get("dbHandler");
    if(theDBHandler==null)
      throw new ParseSubstException("no dbhandler for op "+theOp);
    try{
      theRows=theDBHandler.getQueryRows(theEnv);
    }catch(SQLException ex){
      throw new ParseSubstException("dbHandler for op "+theOp+
```

138

```
                    "failed with SQLException "+ex);
         }
      }
    public void doDef(ParseTree T) throws ParseSubstException{
      Hashtable H=T.getProps(); String name;
      if(H==null || null==(name=(String)H.get("name")))
        throw new ParseSubstException("SUBSTDEF needs a 'name' property");
      StringBuffer valBuff=new StringBuffer();
      outSubVals(T,valBuff);
      theEnv.put(name,valBuff.toString());
      }
    public      void      substStringBuffer(String      tag,ParseTree      T)      throws
ParseSubstException{
        newQuery(T);
      if(tag.equals("myThesis:SUBST")) outSubVals(T);
      else if(tag.equals("myThesis:SUBSTROW")){
        if(substFailure || null==theRows ||
          !theRows.next())
          {substFailure=true; return;}
        else {
          theEnv=theRows.getRow();
          outSubVals(T);
          }
        } else if(tag.equals("myThesis:SUBSTERR")){
          if(substFailure) outSubVals(T);
        } else if(tag.equals("myThesis:SUBSTROWLIST")){
          if(substFailure || null==theRows)return;
          while(theRows.next()){
            theEnv=theRows.getRow();
            outSubVals(T);
            }
        } else if(tag.equals("myThesis:SUBSTDEF"))doDef(T);
        else throw new ParseSubstException("unrecognized tag "+tag);
      }
    public void outSubVals(ParseTree T){
      int N=T.numChildren();
      for(int i=0;i<N;i++)
        if(T.child(i).getTagName().equals("DELIM"))
          outVal(T.child(i));
        else outText(T.child(i));
      }
    public void outText(ParseTree T){
      int low=T.getLow(); int high=T.getHigh();
      if(low>=high)return;
```

```java
    outBuff.append(theString.substring(low,high));
  }
 public void outVal(ParseTree T){
  int low=T.getLow(); int high=T.getHigh();
  if (low>=high)return;
  outBuff.append(theEnv.getStr(theString.substring(low,high)));
 }

 public void outSubVals(ParseTree T,StringBuffer sB){
  int N=T.numChildren();
  for(int i=0;i<N;i++)
   if(T.child(i).getTagName().equals("DELIM"))
     outVal(T.child(i),sB);
   else outText(T.child(i),sB);
 }
 public void outText(ParseTree T,StringBuffer sB){
  int low=T.getLow(); int high=T.getHigh();
  if(low>=high)return;
  sB.append(theString.substring(low,high));
 }
 public void outVal(ParseTree T,StringBuffer sB){
  int low=T.getLow(); int high=T.getHigh();
  if (low>=high)return;
  sB.append(theEnv.getStr(theString.substring(low,high)));
 }
 public String context(int n,int m){
  if(n>=m)return "[no context:"+n+">="+m+"]";
  if(n>=theString.length()) return
    "[no context: "+n+">= string length "+theString.length()+"]";
  if(m>=theString.length())return theString.substring(n,theString.length());
  return theString.substring(n,m);
 }
 public ParseSubst(StringBuffer sB) throws ParseSubstException{
  if(sB==null)throw new ParseSubstException("can't parse null string buffer");
  theBuff=sB;
  theString=sB.toString();
  lex=new MyNaLex(theBuff);
  lg=new Logger();
  theTree=parseRoot();
  substFailure=false;
 }
 public ParseTree parseRoot() throws ParseSubstException{
  ParseTree root=new ParseTree("ROOT",0,theString.length());
  int tokType=lex.getToken();
```

```
      while(MyNaLex.endAllToken!=tokType){
        if(tokType==MyNaLex.noToken)
          throw new ParseSubstException("ERROR: Bad token at parseRoot:\n "
                      +lex.context());
        int lo=lex.getTokenStart(); int hi=lex.getTokenEnd();
        if(tokType==MyNaLex.textToken)
          root.addChild(new ParseTree("TEXT",lo,hi));
        else if(tokType==MyNaLex.mynaToken)
          root.addChild(parseSub(lo,hi));
        else throw new ParseSubstException("ERROR: Bad token at beginning:\n "
                      +lex.context());
        tokType=lex.getToken();
        }
      return root;
    }
    public ParseTree parseSub(int lo, int hi) throws ParseSubstException{
      // the current token is a mynaToken, or it wouldn't be here.
      Hashtable tokProps=lex.getTokenProps();
      String tagName=theString.substring(lo,hi);
      ParseTree pS=new ParseTree(tagName,lo,hi,tokProps);
      int tokType;
      while(MyNaLex.textToken==(tokType=lex.getToken())){
        pS.addChild(new ParseTree("TEXT",
                lex.getTokenStart(),
                lex.getTokenEnd()));
        if(MyNaLex.textToken!=(tokType=lex.getToken()))break;
          // textToken values are TEXT, DELIM alternately.
        pS.addChild(new ParseTree("DELIM",
                lex.getTokenStart(),
                lex.getTokenEnd()));
      }
      String closeTag=null;
      if(tokType!=MyNaLex.endMynaToken
        || !tagName.equals(closeTag=lex.getTokenString()) )
        throw new ParseSubstException("ERROR: <XMP>expecting ["+tagName+
                    "], found ["+closeTag+"]\n"+
                    lex.context()+"</XMP>");
      return pS;
    }
  }
```

APPENDIX U. PARSETREE.JAVA

```java
package mythesis.utilityClasses;
import java.io.*;
import java.util.*;
public class ParseTree {
  int lo=0; int hi=0; String tagName=null;
  Hashtable props=null; Vector children=null;
public ParseTree(String tag){this(tag,0,0,null);}
public ParseTree(String tag, int L){this(tag,L,L,null);}
public ParseTree(String tag, int L,int H){this(tag,L,H,null);}
public ParseTree(String tag, int L,Hashtable H){this(tag,L,L,H);}
public ParseTree(String tag, int L,int H,Hashtable tab){
  tagName=tag;lo=L;hi=H;props=tab;
}
public Object get(Object key){
  if(props==null)return null;
  else return props.get(key);
}
public Object put(Object key,Object val){
  if(props==null)props=new Hashtable(1);
  props.put(key,val);
  return val;
}
public void setTagName(String tag){tagName=tag;}
public void setLow(int i){lo=i;}
public void setHigh(int i){hi=i;}
public String getTagName(){return tagName;}
public int getLow(){return lo;}
public int getHigh(){return hi;}
public void setProps(Hashtable H){props=H;}
public Hashtable getProps(){return props;}
public int numChildren(){
  if(children==null)return 0;
  else return children.size();
}
public ParseTree child(int i){
  if(numChildren()<=i)return null;
  return (ParseTree)children.elementAt(i);
}
public void addChild(ParseTree pT){
  if(children==null)children=new Vector(1);
  children.addElement(pT);
}
}
```

142

APPENDIX V. PROPERTYGROUPS.JAVA

```java
package mythesis.utilityClasses;
import java.util.*;

public class PropertyGroups extends Hashtable {
private Properties props=null;
private String fileName=null;
Logger lg;
int simpleKeys;
int compoundKeys;
public PropertyGroups(){
 props=new Properties();
 lg=new Logger();
 simpleKeys=0;
 compoundKeys=0;
}
public PropertyGroups(String name)throws Exception{
 this();
 load(name);
}
public void load(String name)throws Exception{fileName=name; load();}
public void load()throws Exception{
 java.io.InputStream is=getClass().getResourceAsStream(fileName);
 if(null==is) throw
   new Exception("PropertyGroups.load: can't get resource "+fileName);
 props.load(is);
 is.close();
 Enumeration keys=props.keys();
 while(keys.hasMoreElements()){
  String key=(String)keys.nextElement();
  int sloc;
  if(0>(sloc=key.indexOf('_'))){
    simpleKeys++;
    put(key,props.get(key));
    }
  else{
   String key1=key.substring(0,sloc);
   String key2=key.substring(1+sloc);
   Properties subprops=(Properties)get(key1);
   if(null==subprops){
      compoundKeys++;
      put(key1,subprops=new Properties());
      }
   subprops.put(key2,props.get(key));
```

```java
      } }
    }
    public String getProperty(String key){
      Object ob=get(key);
      if(ob instanceof String)return (String)ob;
      return null;
    }
    public String getProperty(String key,String dflt){
      if(null==key)return dflt;
      String p=getProperty(key);
      return p==null?dflt:p;
    }
    public Properties getProperties(String key){
      if(null==key){lg.logIt("getProperties null"); return null;}
      Object ob=get(key);
      if(ob instanceof Properties)return (Properties)ob;
      return null;
    }
    Enumeration propertyKeys(){return new PropertyKeysEnum();}
    Enumeration simpleKeys(){return new SimpleKeysEnum();}
    private void keyVal(StringBuffer sB,Object key){
      String k=(String)key;
      sB.append(k); sB.append("=");
      sB.append(get(key).toString());
    }
    public synchronized String toString(){
      StringBuffer sB=new StringBuffer();
      Enumeration pk=propertyKeys();
      if(pk.hasMoreElements())keyVal(sB,pk.nextElement());
      while(pk.hasMoreElements()){
        sB.append(", ");
        keyVal(sB,pk.nextElement());
      }
      Enumeration sk=simpleKeys();
      if(sk.hasMoreElements())keyVal(sB,sk.nextElement());
      while(sk.hasMoreElements()){
        sB.append(", ");
        keyVal(sB,sk.nextElement());
      }
      return sB.toString();
    }
    class PropertyKeysEnum implements Enumeration{
      int howMany; Enumeration baseEnum;
    public PropertyKeysEnum(){
```

144

```java
   howMany=compoundKeys;
   baseEnum=keys();
 }
 public boolean hasMoreElements(){return howMany>0;}
 public Object nextElement(){
  Object ob;
  while(baseEnum.hasMoreElements()){
   Object k=baseEnum.nextElement();
   Object v=get(k);
   if(v instanceof Properties){
    howMany--;
    return k;
    }  }
  howMany=0;
  return null;
}} // end of PropertyKeysEnum inner class
class SimpleKeysEnum implements Enumeration{
  int howMany; Enumeration baseEnum;
 public SimpleKeysEnum(){
  howMany=simpleKeys;
  baseEnum=keys();
 }
 public boolean hasMoreElements(){return howMany>0;}
 public Object nextElement(){
  Object ob;
  while(baseEnum.hasMoreElements()){
   Object k=baseEnum.nextElement();
   Object v=get(k);
   if(v instanceof String){
    howMany--;
    return k;
    }  }
  howMany=0;
  return null;
}} // end of SimpleKeysEnum inner class
}
```

```java
package mythesis.utilityClasses;
import java.sql.*;
import java.util.Hashtable;
/* the RowSequence provides access to a ResultSet as
an sequence of Env (extended Hashtable) structures:
with RowSequence rS,  rS.next() will  shift to the next Env and return true,
reading and processing a row -- or fail.
and rS.getRow() will return the current Env.
*/
public class RowSequence {
  ResultSet theResultSet;
  Env theEnv;// holds query, resultset and current row info.
  String [] theColumnLabels;
  String [] theColumnTypes;
  String [] theColumnValues;
  int theNumberOfColumns;
 public RowSequence(ResultSet R,Env queryInfo)
    throws SQLException{
  theResultSet=R;
  theEnv=queryInfo;
  if(R==null){
   theColumnLabels=null;
   theColumnTypes=null;
   theColumnValues=null;
   theNumberOfColumns=0;
  }
  // extract informtaion from ResultSet using MiscDB utilities.
   else{
  theColumnLabels=MiscDB.resultSetLabels(R);
  theColumnTypes=MiscDB.resultSetTypes(R);
  theNumberOfColumns=theColumnLabels.length;
  theColumnValues=new String[theNumberOfColumns];
  theEnv.put("NumberOfColumns",""+theNumberOfColumns);
  theEnv.put("FieldName",theColumnLabels);
  theEnv.put("FieldType",theColumnTypes);
  theEnv.put("FieldValue",theColumnValues);
  for(int i=1;i<=theNumberOfColumns;i++)
   theEnv.put("FieldName"+i,theColumnLabels[i-1]);
  }
 }
 public RowSequence(ResultSet R)
    throws SQLException{
```

```java
    this(R,new Env());
   }
  // gets values from the result set row and store in Env
  public boolean next(){
    if(theResultSet==null)return false;
    try{
     if(!theResultSet.next()){
       theResultSet.close();
       theResultSet=null;
       return false;
       }
     for(int i=1;i<=theNumberOfColumns;i++){
       String S=theResultSet.getString(i);
       theColumnValues[i-1]=S;
       theEnv.put(theColumnLabels[i-1],S);
       theEnv.put("FieldValue"+i,S);
       }
     return true;
   }catch(Exception E){}
    return false;
  }
  public Env getRow(){
    return theEnv;
  }
  // tools for inheritance, e.g. pruning a RowSequence.
  public void initFromRowSequence(RowSequence re){ // a shallow copy
    this.theResultSet=re.theResultSet;
    this.theEnv=re.theEnv;
    this.theColumnLabels=re.theColumnLabels;
    this.theColumnTypes=re.theColumnTypes;
    this.theColumnValues=re.theColumnValues;
    this.theNumberOfColumns=re.theNumberOfColumns;
  }
  public RowSequence(){} // a do-nothing empty constructor

}
```

APPENDIX X. ROWSUBST.JAVA

```java
package mythesis.utilityClasses;
import java.io.BufferedReader;
import java.io.PrintWriter;
import java.io.FileReader;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.util.Hashtable;
import java.util.Enumeration;

public class RowSubst {
  RowSequence theRows; Env theInitEnv; Logger lg;
  String theFileName;   BufferedReader theInput;
  PrintWriter theOutput; boolean dataFailed;
  String theCommandPrefix="$$SUBST";
  String SUBST="";
  String SUBSTD="D";
  String SUBSTDERR="DERR";
  String SUBSTDSTAR="D*";
public RowSubst(String fName,RowSequence rows,PrintWriter out) {
  this(newReader(fName),rows,out);
  theFileName=fName;
}
public static BufferedReader newReader(String fName){
  BufferedReader bR;
  try{bR=new BufferedReader(new FileReader(fName));}
  catch(FileNotFoundException Ex){return null;}
  return bR;
}
public RowSubst(BufferedReader brin,RowSequence rows,PrintWriter out) {
  theInput=brin; theOutput=out; theRows=rows;
  theInitEnv=rows.getRow();
  dataFailed=false;
  }
public String interpretLine(String L){
  if(L==null)return "";
  if(!L.startsWith(theCommandPrefix))return L;
  try{
    int cmdBegin=theCommandPrefix.length();
    int cmdEnd=L.indexOf(':');
    int delimEnd=L.indexOf(':',1+cmdEnd);
    String cmd=L.substring(cmdBegin,cmdEnd);
    String delim=L.substring(1+cmdEnd,delimEnd);
    L=L.substring(1+delimEnd,L.length());
```

```java
      if(SUBST.equals(cmd)){
        return Misc.stringDelimSubst(L,delim,theInitEnv);
      } else if (SUBSTD.equals(cmd)){
        if(theRows.next())
          return Misc.stringDelimSubst(L,delim,theRows.getRow());
        else {dataFailed=true; return "";}
      } else if (SUBSTDERR.equals(cmd)){
        if(!dataFailed)return "";
        else return Misc.stringDelimSubst(L,delim,theInitEnv);
      }else if (SUBSTDSTAR.equals(cmd)){
        String S="";
        while(theRows.next())
          S+=Misc.stringDelimSubst(L,delim,theRows.getRow())+"\n";
        return S;
      } else return L;
      }catch(Exception E){}
    return L;
  }
  public void interpret(){
    String nextLine;
    if(theInput==null)return;
    try{
    while(null!=(nextLine=theInput.readLine()))
      theOutput.println(interpretLine(nextLine));
    theInput.close();
    }catch(IOException e)
        {try{theInput.close();}catch(IOException ex){}}
  }
}
```

APPENDIX Y. CACHE.JAVA

```java
package mythesis.utilityClasses;
import java.util.*;
public class Cache {

public static synchronized Cache getInstance(){return null;}
public static synchronized int freeInstance(){return 0;}
public static synchronized boolean close()throws Exception{return false;}
Logger lg;
protected Cachetable cache=null;
protected Cache(){init();}    // constructor only called internally
protected void init(){
  cache=new Cachetable();
```

149

```
      lg=new Logger();
  } // set up cache; can register with a CacheRegistry here.
  public boolean freeItem(Object ob){
    return true;
    // called by freeSpace; override if you need to do anything here.
  }
  public int freeSpace(int numToFree){
    return cache.freeSpace(numToFree);
  }
  public Object get(Object k){
  return cache.get(k);
  }
  public Object get(Object k1,Object k2){
    return cache.get(k1,k2);
  }
  public Object get(Object k1,Object k2,Object k3){
    return cache.get(k1,k2,k3);
  }
  public Object put(Object k,Object v){
    cache.put(k,v);
    return v;
  }
  public Object put(Object k1,Object k2,Object v){
   cache.put(k1,k2,v);
    return v;
  }
  public Object put(Object k1,Object k2,Object k3,Object v){
   cache.put(k1,k2,k3,v);
    return v;
  }
class Cachetable extends Hashtable {
public Cachetable(){ }
public synchronized Object put(Object k,Object v){
  if(null==k)k="";
  return super.put(k,v);
}
public synchronized void put(Object k,Object k2,Object v){
  Cachetable subCache=(Cachetable)get(k);
  if(null==subCache)put(k,subCache=new Cachetable());
  subCache.put(k2,v);
}
public synchronized void put(Object k,Object k2,Object k3,Object v){
  Cachetable subCache=(Cachetable)get(k);
  if(null==subCache)put(k,subCache=new Cachetable());
```

```java
      subCache.put(k2,k3,v);
    }
public synchronized Object get(Object k){
  return super.get(null==k?"":k);
}
public synchronized Object get(Object k,Object k2){
  Cachetable subCache=(Cachetable)get(k);
  if(null==subCache)return null;
  return subCache.get(k2);
}
public synchronized Object get(Object k,Object k2,Object k3){
  Cachetable subCache=(Cachetable)get(k);
  if(null==subCache)return null;
  return subCache.get(k2,k3);
}
public synchronized int freeSpace(int numToFree){ // returns leftover.
  Enumeration enum=keys();
  while(enum.hasMoreElements() && numToFree>0){
    Object k=enum.nextElement();
    Object v=get(k);
    if(v instanceof Cachetable){
      Cachetable sub=(Cachetable)v;
      numToFree=sub.freeSpace(numToFree);
      if(numToFree>0)remove(k); // didn't free enough, must be empty.
    }else {
      freeItem(v);
      remove(k);
      numToFree--;
    }
  }
  return numToFree; // this many still not freed.
}
} // end of Cachetable inner class
}
```

APPENDIX Z. SMART DATABASE RELATIONSHIP DIAGRAM

LIST OF REFERENCES

1. Orfali, R, Harkey, D. Edwards, J., The Essential Client/Server Survival Guide, Wiley, 1996.

2. Buczek, Greg, Instant ASP Scripts, McGraw – Hill, 1999.

3. Eckerson, Wayne W., Three-Tier Client/Server Architecture, Open Information Systems, 1995.

4. Schussel, George, Client/Server Past Present and Future, Online, Internet, Avalible, http://news.dci.com/geos/dbsejava.htm

5. Deitel, Harvey, M., Internet & World Wide Web How To Program, Prentice Hall, 1999.

6. Hunter, Jason, Java Servlet Programmimg, O'Reilly 1998.

7. Moss, Karl, Java Servlets, Second Edition, McGraw – Hill, 1999.

8. Deitel, Harvey, M., Deitel, Paul, S., Java How To Program, Prentice Hall, 1999.

9. Slender, Grant, ASP vs JSP Test Page, Online, Internet, Avalible, http://www.geocities.com/gslender

10. Walther, Stephen, Banick, Steve, Active Server Pages 2.0 Unleashed, Sams Publishing, 1999.

11. Patzer, Andrew, Li, Sing, Professional Java Server Programming, Wrox Press, 1999.

12. Nakhimovsky, Alexander, Myers, Tom, Professional Java XML Programming with Servlets and JSP, Wrox Press, 1999.

13. Bergsten, Hans, Connection Pooling with Java Servlets, Online, Internet, Avalible http://news.dci.com/geos/dbsejava.htm

14. Hall, Marty, Core Servlets And Java Server Pages, Prentice Hall, 2000.

15. George, Reese, Java Database Programming with JDBC and Java, O'Reilly 1997.

16.   Fisher, Maydene,   JDBC Database Access Online, Internet, Available http://java.sun.com/docs/books/tutorial/jdbc/index.html.

17.   Ablan, Jery, Developing Intranet Applications with Java, Sams Publishing 1996.

18.   Orfali, R, Harkey, D., Client/ Server Programming With Java and Corba, Second Edition, John Wiley and Sons Inc., 1998.

19.   Kroenke, D., Database Processing Fundamentals, Design And Implementation, Sixth Edition., Prentice Hall 1999.

20.   Java Server Pages White Paper 14 Feb 2001, Online, Internet, Available, http://java.sun.com/products/jsp/whitepaper.html

INITIAL DISTRIBUTION LIST